Fault-Tolerant Many-Cores for Mixed-Critical Real-Time Systems

Von der Fakultät für Elektrotechnik, Informationstechnik, Physik
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines Doktors

der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von Eberle Andrey Rambo
aus Itajaí, Brasilien

Eingereicht am: 06.11.2018
Mündliche Prüfung am: 18.01.2019

1. Referent: Prof. Dr.-Ing. Rolf Ernst
2. Referent: Dr.-Ing. Leandro Soares Indrusiak

Druckjahr: 2019

We are what we believe we are.

– *C.S. Lewis*

Set your life on fire. Seek those who fan your flames.

– *Rumi*

Abstract

Technology downscaling has increased the hardware's overall susceptibility to random hardware faults to the point where they became non-negligible. Hence, current and future computing systems must be appropriately designed to cope with errors in order to provide a dependable service and correct functionality. Dependability has many facets to be addressed when designing a system and that is specially challenging in mixed-critical real-time systems, where safety standards play an important role and where responding in time can be as important as responding correctly or even responding at all. Moreover, future mixed-critical multi- and many-core platforms requires sufficient independence among the applications with different requirements and criticalities that co-exist in the system, further intensifying the challenge.

Cross-layer fault-tolerance solutions are the key to effectively and efficiently achieve dependability in future mixed-critical real-time systems. The thesis addresses the dependability of mixed-critical real-time systems, considering three important requirements: integrity, resilience and real-time. More specifically, it looks into the architectural and performance aspects of achieving dependability, concentrating its scope on error detection and handling in hardware – more specifically in the Network-on-Chip (NoC), the backbone of modern MPSoC – and on the performance of error handling and recovery in software.

The thesis starts by looking at the impacts of random hardware faults on the NoC and on the system, with special focus on soft errors. Then, it addresses the uncovered weaknesses in the NoC by proposing a resilient NoC for mixed-critical real-time systems that is able to provide a highly reliable service with transparent protection for the applications. With a strong fault containment and a resilient router design, the proposed approach limits the impacts of errors in time and in space, resulting in a NoC that is highly available but lossy under errors. The reliable transport of data is then achieved in the upper layers of the network stack by ARQ-based protocols with real-time guarantees. The formal communica-

tion time analysis of common ARQ protocols is modeled for NoCs and includes a novel ARQ-based protocol optimized for DMAs.

After addressing the efficient use of ARQ-based protocols in NoCs, the thesis proposes the Advanced Integrity Q-service (AIQ) approach. AIQ is a low-overhead mechanism to achieve integrity and real-time guarantees of NoC transactions on an End-to-End (E2E) basis. Inspired by transactions in distributed systems, the mechanism differs from the previous approach in that it does not provide error recovery in hardware but delegates the task to software, making use of existing functionality in cross-layer fault-tolerance solutions.

Finally, the thesis addresses error handling in software as seen in cross-layer approaches. It addresses the performance of replicated software execution in many-core platforms. Replicated software execution provides protection to the system against random hardware faults. It relies on hardware-supported error detection and error handling in software. The replica-aware co-scheduling is proposed to achieve high performance with replicated execution, which is not possible with standard real-time schedulers.

Kurzfassung

Die fortschreitende Miniaturisierung der Halbleitertechnik hat die Anfälligkeit von Hardware gegenüber zufälligen Fehlern so weit erhöht, dass sie nicht mehr vernachlässigbar sind. Um einen zuverlässigen Betrieb und korrekte Funktionalität zu gewährleisten, müssen aktuelle und zukünftige Computersysteme so ausgelegt werden, dass sie mit diesen Fehlern umgehen können. Zuverlässigkeit hat viele Aspekte, die bei der Entwicklung eines Systems berücksichtigt werden müssen. Das gilt insbesondere für Echtzeitsysteme mit gemischter Kritikalität, bei denen Sicherheitsstandards, die ein korrektes und rechtzeitiges Verhalten fordern, eine wichtige Rolle spielen. Darüber hinaus erfordern zukünftige Multi- und Many-Core-Plattformen eine ausreichende Unabhängigkeit zwischen Anwendungen mit unterschiedlichen Anforderungen und Kritikalitäten, die im System nebeneinander existieren, wodurch die Herausforderung weiter erhöht wird.

Cross-Layer-Fehlertoleranzlösungen sind der Schlüssel, um effektiv und effizient die Zuverlässigkeit von zukünftigen gemischt-kritischen Echtzeitsystemen zu erreichen. Diese Dissertation befasst sich mit der Zuverlässigkeit von gemischt-kritischen Echtzeitsystemen unter Berücksichtigung von drei wichtigen Anforderungen: Integrität, Resilienz und Echtzeit. Genauer gesagt, behandelt sie Architektur- und Leistungsaspekte die notwendig sind um Zuverlässigkeit zu erreichen, wobei der Schwerpunkt auf der Fehlererkennung und -behandlung in der Hardware – genauer gesagt im Network-on-Chip (NoC), dem Rückgrat des modernen MPSoC – und auf der Leistung der Fehlerbehandlung und -behebung in der Software liegt.

Die Arbeit beginnt mit der Untersuchung der Auswirkung von zufälligen Hardwarefehlern auf das NoC und das System, wobei der Schwerpunkt auf weichen Fehler (soft errors) liegt. Anschließend werden die aufgedeckten Schwachstellen im NoC behoben, indem ein widerstandsfähiges NoC für gemischt-kritische Echtzeitsysteme vorgeschlagen wird, das in der Lage ist, einen höchst zuverlässigen Betrieb mit transparentem Schutz für die Anwendungen zu bieten. Mit einer starken Fehlerbegrenzung und einem robusten Router-Design begrenzt die vorgeschlagene Methode die Auswirkun-

gen von Fehlern in Zeit und Raum, was zu einem hochverfügbaren NoC führt, dass aber verlustbehaftet bei auftretenden Fehlern ist. Der zuverlässige Transport der Daten erfolgt in den oberen Schichten des Netzwerkstacks durch ARQ-basierte Protokolle mit Echtzeitgarantie. Die formale Kommunikationszeitanalyse gängiger ARQ-Protokolle ist für NoCs modelliert und beinhaltet ein neuartiges, für DMAs optimiertes ARQ-basiertes Protokoll.

Nach der Auseinandersetzung mit der effizienten Nutzung von ARQ-basierten Protokolle in NoCs, wird die Advanced Integrity Q-Service (AIQ) Methode vorgestellt. AIQ ist ein Mechanismus mit geringem Overhead, um Integrität und Echtzeit-Garantien von NoC-Transaktionen auf Ende-zu-Ende (E2E)-Basis zu erreichen. Inspiriert von Transaktionen in verteilten Systemen unterscheidet sich der Mechanismus vom bisherigen Konzept dadurch, dass er keine Fehlerbehebung in der Hardware vorsieht, sondern diese Aufgabe an die Software delegiert und dabei die vorhandene Funktionalität in schichtübergreifenden Fehlertoleranzlösungen nutzt.

Schließlich befasst sich die Dissertation mit der Fehlerbehandlung in Software, wie sie in schichtübergreifenden Methoden zu sehen ist. Sie behandelt die Leistung der replizierten Software-Ausführung in Many-Core-Plattformen. Replizierte Software-Ausführung schützt das System vor zufälligen Hardwarefehlern. Es setzt auf hardwaregestützte Fehlererkennung und Fehlerbehandlung in der Software. Das Replika-bewusste Co-Scheduling wird vorgeschlagen, um eine hohe Performance bei replizierter Ausführung zu erreichen, was mit Standard-Echtzeit-Schedulern nicht möglich ist.

Contents

1. Introduction

Embedded systems are ubiquitous nowadays. Seen or unseen, they are everywhere, from kitchen appliances and healthcare gadgets to medical and military devices. It is extremely difficult to picture a life in modern days without daily contact with an embedded system, except due to extreme lifestyles or due to the lack of choice. Embedded systems are information processing systems embedded into enclosing products [97]. A real-time embedded system is one that additionally must react within precise time constraints to events in the environment [24], in other words, where actions are constrained in time.

Real-time embedded systems can be divided into soft and hard real-time systems depending on how serious it is to miss a task deadline, i.e., to violate its *timing constraints* [47, 93]. A timing constraint is hard when the consequence of missing a deadline is fatal, a late response is useless and unacceptable. A timing constraint is soft when the consequence of missing a deadline is undesirable but tolerable. A soft real-time system is then a system with only soft timing constraints. A hard real-time system is a system with at least one hard timing constraint and may also contain soft timing constraints.

Systems are safety relevant when they control or at least potentially impact the functional safety of any application [46]. Thus, hard real-time embedded systems are almost always also *safety-critical* systems. The inverse is always true – i.e., a *safety-critical* system is a hard real-time system. Examples of applications are the Flight Management System (FMS), Advanced Driver Assistance System (ADAS) and autonomous driving, in the aerospace and automotive industries, respectively. Due to the impacts of a failure, such systems are required to be certified, proving that the processing architecture is capable of meeting the specified safety goals. The certification processes and goals are defined in standards such as the IEC 61508 [69] and the domain-specific counter-parts ISO 26262, for automotive electric/electronic systems [72], and DO-254, for airborne electronic hardware [35]. The standards also define up to five criticality levels

called Safety Integrity Levels (SILs), Automotive Safety and Integrity Levels (ASILs) and Design Assurance Levels (DALs), respectively. The proof usually requires a Failure Mode and Effects Analysis (FMEA) [68], which systematically captures all potential faults and their effects, which must be appropriately addressed.

Demanded by the market, increasingly complex new functionalities are being implemented by such safety-critical systems nowadays. Automotive vehicles and aircrafts contain a number of safety-relevant functions and many others that are not, the so-called comfort functions [46]. In order to meet stringent non-functional requirements w.r.t. cost, space, weight and power consumption, such systems are evolving into complex *mixed-critical* systems where components with different criticality levels are integrated into a common hardware platform [22]. That raises yet another crucial safety requirement, the *sufficient independence* [72], which requires a function not to be affected by the misbehavior or failure of another function in the system. Moreover, complex functionalities of future systems require the appropriate computational power and high speed communication to process the large amounts of information and images of demanding applications such as FMS and autonomous driving. Those factors have contributed to a migration from single-core platforms to multi-cores and to future many-core architectures [103, 22].

1.1. Multi- and many-core real-time systems

Multi- and many-core processors have been widely adopted by the server and consumer electronics markets. They present better performance and are more efficient with respect to power, area and cost than single-cores. Such processors are currently being evaluated for embedded markets as real-time safety-critical systems [103]. Not only do they allow the integration of multiple applications in a single chip but they also provide the performance required to implement the increasingly complex functionalities demanded by the market. Nonetheless, the migration from single to multi and many-cores in the the real-time domain is complex and has been the subject of research for years. As pointed out by Burns and Davis, the fundamental research question has been "reconciling the conflicting requirements of *partitioning* for (safety) assurance and *sharing* for efficient resource usage". That has given rise to research platforms that investigate

a range of techniques and mechanisms specifically for the mixed-critical real-time domain [46, 14], such as CompSOC [54] and the Integrated Dependable Architecture for Many-Cores (IDAMC) [103, 150].

The IDAMC is a research platform developed at the Institute of Computer and Network Engineering at TU Braunschweig, Braunschweig, Germany. IDAMC has been used to prototype and evaluate several techniques and mechanisms addressing such as predictable memory access scheduling [42], shared resource management [84], and Quality-of-Service (QoS) of on-Chip traffic [32]. This platform has been widely used as a starting point for the experiments reported in this thesis. Figure 1.1 gives an overview of IDAMC. The many-core platform features up to 64 *nodes*. Each *node* is composed of a *router* and up to 4 *tiles*, which can be, e.g., a processing element, hardware accelerator or a memory controller. The routers are connected to each other forming a network, a Network-on-Chip (NoC), and are connected to the tiles with *network interfaces*.

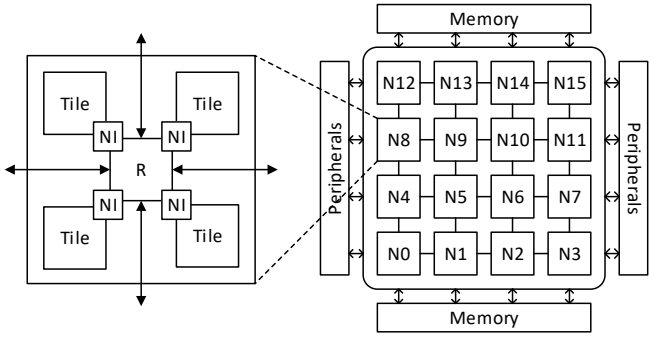


Figure 1.1.: Overview of the IDAMC many-core platform.

Many-core platforms have also been commercially available for some time. They target high-throughput applications such as networking, multimedia and storage. Although not widely used in production of safety-critical real-time systems, companies are actively involved in research towards meeting the stringent requirements of those systems. Examples are Kalray's Massively Parallel Processor Array (MPPA)® many-core processor family [77] and Mellanox's TILE-Gx36™ and TILE-Gx72™ [100]. As an illustrative example, the processor block diagram of TILE-Gx8072™ is shown in Figure 1.2. The processor features 72 64-bit cores, each with

split 32+32KB L1 caches, one 256KB L2 cache, and a maximum frequency of 1.2GHz. A shared, coherent 18MB L3 cache is also included along with 4 DDR3 controllers. For networking and system integration, the chip includes USB, I2C, SPI, UART and JTAG interfaces as well as 10Gbps XAUI and SGMII ports. The cores, interfaces and ports are connected by an iMesh interconnect consisting of five independent low-latency NoCs.

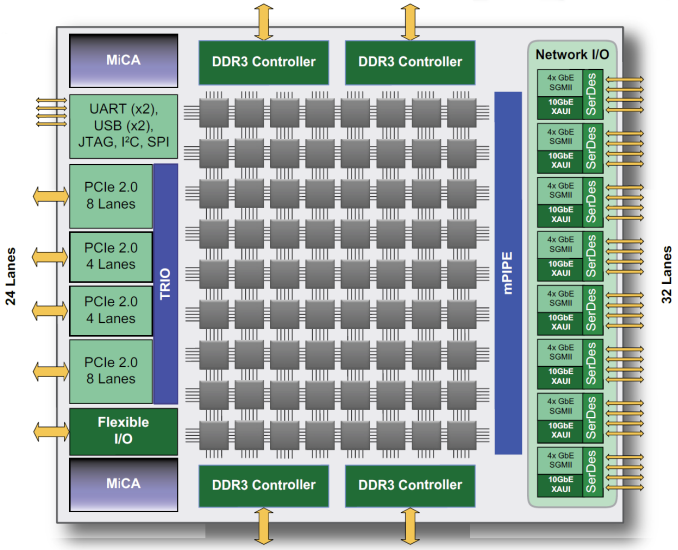


Figure 1.2.: TILE-Gx8072^{TM} processor block diagram [100].

1.2. Network-on-Chip

To handle the high-bandwidth requirements of such complex platforms, the NoC was developed as the scalable interconnect solution [15]. NoCs overcame the limitations of buses, which quickly became the bottleneck as designs grew in size and complexity, and they are the interconnect to be found in modern multi- and many-core processors and Systems-on-Chip (SoCs) [15]. NoCs provides for easier wire routing and easier integration of heterogeneous components in modern, large and complex Multiprocessor System-on-Chip (MPSoC) designs [4]. As an important shared component, the

NoC concentrates all communication entering, leaving or occurring within the chip. That includes, e.g., all I/O operations and memory transactions accessing a memory external to the tiles. Many different types of NoCs can be found in academia as well as available commercially. Examples of commercial NoCs are Arteris' FlexNoC® [4] and Ncore™ [5], Netspeed's Orion [108] and Gemini [107], and Sonics' family of NoC Intellectual Properties (IPs) [143]. The work reported in this thesis has been largely based on the IDA NoC, featured in the research platform IDAMC.

The IDA NoC is a NoC for real-time mixed-critical systems [103]. The network implements XY deterministic *source routing*, where the route and Virtual Channel (VC) are defined at the source; *wormhole switching*, where variable-sized packets are composed of fixed-sized Flow Control Units (flits); and *VC flow control*, where flits transit through a number of VCs. The routers implements the *SLIP arbitration* [99], a two-stage round-robin scheduler.

The block diagram of the IDA NoC router is depicted in Figure 1.3. The routers are input-buffered and the input buffer contains a FIFO queue for each VC. The switch arbiter implements the SLIP arbitration. The Virtual Channel Access Controller (VCAC) manages the access to VCs in downstream routers. The crossbar switch connects the input buffers to the respective output ports according to the arbitration grants. The flits are then forwarded from the input buffer directly to the downstream router. The router implements stop-and-wait flow control, raising a stop signal to inform the near-full state of the VC queues to the upstream router.

Route management is performed before the flit is stored in buffers to keep the routing data for the next router always in the first position. The route is encoded as a list of *runs*, which is a pair: output port (direction) and number of hops (distance). During a run, the hop counter is decremented. Once the packet finishes a run, the route is rotated: the finished run is moved to the end of the list and the next run becomes the head. The router execution spans a 4-stage pipeline.

The packets have variable size and are composed of a Head Flit (HF), zero or more Body Flits (BFs), and one Tail Flit (TF). A Single Flit (SF) packet is also supported. The flit formats are depicted in Figure 1.4. HFs and SFs (resp. BFs and TFs) are identical except for the flit type, which distinguishes their semantics. For transmission, a flit may be further divided in Physical Units (phits) to match the pipeline depth – i.e., 4 phits

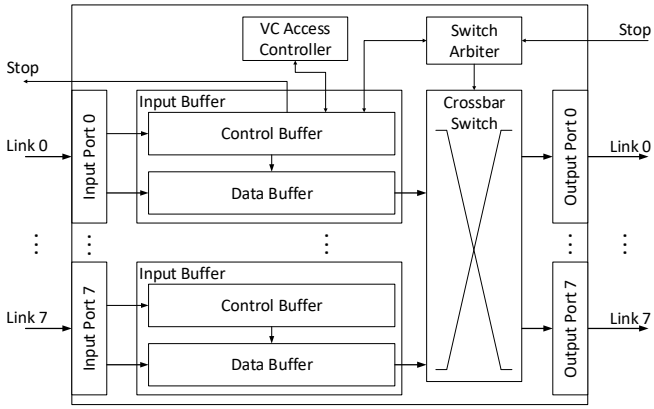


Figure 1.3.: Overview of the IDA NoC router.

HF/SF	VC	Flit Type	Route	Tile Port	Payload
Size	3 bits	2	30	3	102
BF/TF	VC	Flit Type	Payload		
Size	3 bits	2	135		

Figure 1.4.: Single (SF), Head (HF), Body (BF) and Tail (TF) flit formats.

of 35 bits, not shown in Figure 1.4.

1.3. Dependability challenge

Besides predictability and efficient resource usage, mixed-critical real-time systems face yet another challenge: the *dependability* challenge. Not only must the system be operating correctly under ideal conditions but must also continue operating under less-than-ideal conditions. At the same time that technology downscaling has enabled many-core chips with billions of transistors, it has also increased the overall system susceptibility to hardware errors to a point where they cannot be neglected in the higher levels of abstraction anymore [20, 41, 52, 61, 64]. Thus, real-time systems must ensure not only that the timing constraints are met but also that the de-

pendability requirements are met, such as defined by standards IEC 61508 [69], ISO 26262 [72] and DO-254 [35].

The possible *attributes* of a dependable system are: *availability*, *reliability*, *safety*, *integrity* and *maintainability* [7]. According with Avizienis et al., availability is the readiness for correct service; reliability is the continuity of correct service; safety is the absence of catastrophic consequences on users and environment; integrity is the absence of improper system alterations; and maintainability is the ability to undergo modifications and repairs. Alternatively, safety is defined by the ISO 26262 standard as the absence of unreasonable risk [72].

The reliability of components and systems[1] is usually quantified by two important metrics: Mean Time To Failure (MTTF) and Failures in Time (FIT) [112]. The MTTF is the arithmetic mean of the time it takes for a non-repairable system to fail, usually expressed in hours [67]. The MTTF can be obtained as follows [67]:

$$MTTF = \int_0^\infty \mathcal{R}(t)\ dt \tag{1.1}$$

where $\mathcal{R}(t)$ is the reliability in time, also called the survivor function. It can be expressed as follows[2] [67]:

$$\mathcal{R}(t) = e^{-\lambda \cdot t} \tag{1.2}$$

where λ is the constant failure rate. The FIT metric measures the number of failures in a billion hours of device-hour operation [67]. FIT can obtained from the MTTF, and vice versa, when the latter is given in hours:

$$FIT = \frac{10^9}{MTTF} \tag{1.3}$$

Other metrics are also employed, specially in very specific domains, but are not as relevant as MTTF and FIT, which can also be employed across domains.

In the sequel, the *threats* to dependability and its attributes (faults, errors and failures) are discussed. Afterwards, the *means* to attain dependability in spite of the threats are discussed.

[1] A system is a set of components [72].

[2] Considering an exponential distribution of failures with a constant failure rate λ, which models well the failures in the useful lifetime of a system [67].

1.4. Threats to dependability: random hardware faults

The *threats* to dependability and its attributes are *faults*, *errors* and *failures* [7]. According to Avizienis et al., a fault is the cause of an error; the error is a deviation of the external state of the system from the correct service; when not handled, an error causes a failure; a failure is then an event that occurs when the delivered service deviates from the correct service. Similarly, ISO 26262 [72] defines a fault as an abnormal condition that can cause the failure of the system; error is a discrepancy between computed (or observed, measured) value (or condition) and the true (or specified, theoretically correct) value (or condition); and failure is the termination of the ability of the system to perform a function as required.

Errors can be caused essentially by faults of three groups regarding their sources [7]:

- Development faults: occurring during the development (design flaws);
- Physical faults: affecting the hardware; and
- Interaction faults: external faults.

This thesis focuses on physical faults, which are also called *random hardware faults* by ISO 26262 [72]. It is therefore assumed the absence of development and interaction faults, also called systematic faults [72], which can be achieved by good practices and processes, and design for safety.

Random hardware faults cause errors that can be classified in two types [52]:

- Soft errors: caused by transient and intermittent faults, they are events where the data is corrupted (*bit-flip*) and the device is not permanently damaged.
  - A transient fault occurs once and subsequently disappears; it is caused, e.g., by electromagnetic interference and Single-event effects (SEEs).
  - An intermittent fault occurs time and time again, then disappears usually in a component that is on the verge of breaking down due to wear out.

  Detailed classification of soft errors is introduced in Chapter 2.
- Hard errors: caused by permanent faults, they are events where the

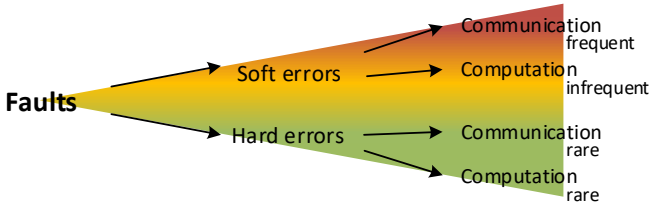


Figure 1.5.: Occurrence probability of random hardware faults: computation vs. communication.

device fails permanently – e.g., a broken network cable due to mechanical destruction or a stuck-at fault due to aging.

Different types of faults have different occurrence probability and impacts on different parts of the system. Efficacy and efficiency, therefore, require them to be tackled with different strategies. Figure 1.5 decomposes and illustrates the impacts of random hardware faults. Soft errors are by far the most frequent, occurring orders of magnitude more frequently than hard ones. By separating soft errors in computation and communication, one sees that the latter are frequent but not as frequent as the former. In fact, they are orders of magnitude more frequent. For instance, in automotive Ethernet [21] transmissions are guaranteed to have at most one error (bit flip) in 10^{10} transmitted bits – i.e., a Bit Error Rate (BER) lower than or equal to 10^{-10}. In computation, on the other hand, expected[3] BERs are in the order of 10^{-12} bit-flips per hour [6]. The design must however consider higher rates (up to 10^{-9}/hour) as a safety margin [91]. Hard errors are relatively infrequent in both computation and communication. They usually become perceptible and of importance in systems for high dependability after soft errors are appropriately handled. Once the system is able to tolerate soft errors, the next limitation in achieving high dependability are the hard errors.

Random hardware faults are an ever increasing concern in electronic as well as in system design. Ebrahimi et al. showed that error rates per bit are decreasing with technology scaling. That contradicts early predictions [20] that have not been confirmed thanks to improvements in Electronic design automation (EDA) tools and in the device [41, 88]. Nonetheless,

[3]BERs derived for sequential and combinational logic [61] with data from [6] for 65nm CMOS SRAM. Masking effects [41] are not taken into account.

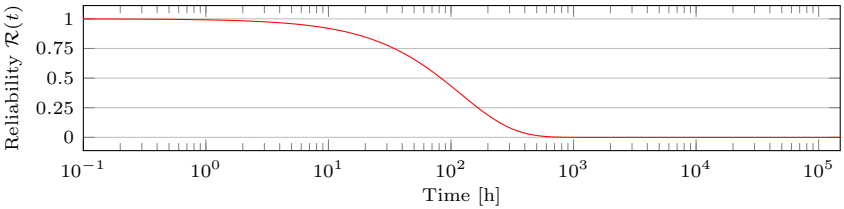


Figure 1.6.: Reliability of unprotected system.

the size and complexity of designs have been increasing much faster than the error rates decrease, which has resulted in an overall increased vulnerability. Interestingly, it was also shown that the Soft Error Rate (SER) of combinatorial logic increases with the frequency whereas the SER of sequential logic decreases with the frequency [41].

To illustrate the importance of such faults in the system, assume a safety-critical system under development, which must meet a certain reliability requirement. The first step is to assess its reliability, e.g., by means of its MTTF. Assume an initially unprotected system consisting of 4 processors, a memory and a NoC. The exponential distribution is chosen to model the reliability of the chip during its useful life period [67]. Constant failure rates due to soft errors $\lambda_p = 3 \cdot 10^{-6}$, $\lambda_m = 8.35 \cdot 10^{-3}$ and $\lambda_n = 3 \cdot 10^{-6}$ are assumed for each processor, memory and NoC, respectively. Additionally, a failure rate due to hard errors $\lambda_h = 10^{-7}$ is assumed for the entire system. The overall system failure rate can be obtained:

$$\begin{aligned} \lambda &= 1 - \left[(1-\lambda_p)^4 \cdot (1-\lambda_m) \cdot (1-\lambda_n) \cdot (1-\lambda_h)\right] \\ &= 8.37 \cdot 10^{-3} \end{aligned} \tag{1.4}$$

The reliability $\mathcal{R}(t)$ is then obtained with Equation 1.2 and the failure rate λ. The resulting reliability function is better visualized in time in the plot of Figure 1.6. After resolving the integral of Equation 1.1, a MTTF of the system of 119 hours is obtained. In terms of FITs, obtained with Equation 1.3, that corresponds to $8.36 \cdot 10^6$ FITs. That is far from the reliability requirements of safety-critical systems, which usually lie below 100 FITs for less critical systems and below 1 FIT for highly-critical ones [72, 69].

1.5. Attaining dependability

Dependability in a system can be *attained* by different *means*. According with Avizienis et al., the means can be grouped in four major categories: *fault prevention*, *fault tolerance*, *fault removal* and *fault forecasting*. Fault prevention refers to prevention by using appropriate hardware processes and hardware and software development methodologies. Fault tolerance is carried out via error detection and recovery, which involves many techniques such as fault masking, rollback, rollforward, compensation, diagnosis, isolation, reconfiguration, and reinitialization. Error detection can be carried out preemptively or concurrently – i.e., while activity system is inactive or during operation. Fault removal refers to the removal of faults during development through validation and test, and during use through corrective and proactive maintenance. Fault forecasting refers to qualitative and quantitative evaluations of the system behavior. This thesis focuses on the synergetic application of fault prevention, tolerance and forecasting techniques

In order to assess the vulnerabilities of the system, qualitative and quantitative evaluations of fault forecasting techniques are crucial. Only after sufficient knowledge on the relevant threats and its impacts on the system and its users, a meaningful and efficient strategy can be formulated. In practice, such assessments usually involve an FMEA or an Failure Mode, Effects and Criticality Analysis (FMECA) [67, 68], as seen in ISO 26262 [72]. The FMEA systematically captures, in a bottom-up approach, all potential faults and their effects by scrutinizing each system component for its impact, when faulty, on the system behavior. The FMECA extends the FMEA by assigning criticalities or priorities to effects [67, 68]. Naturally, such evaluations are also imperative at the end of the project as means to prove that the required dependability was attained [72]. That is usually required in certification processes of safety standards [69, 72, 35]. More details in Chapter 2.

Fault tolerance can then be implemented in a variety of ways that incur different cost and performance penalty in the system under design. Fault-tolerant approaches can be classified with respect to the abstraction level where it handles faults and errors:

- Component-level fault-tolerance approaches;
- System-level fault-tolerance approaches; or

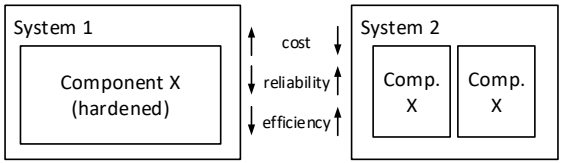


Figure 1.7.: Example of two instances of reliable systems: the case for cross-layer solutions.

- A combination of both in a *cross-layer* approach.

Cross-layer approaches are particularly interesting because they enable a larger design space where the cost-reliability trade-off can be exploited with different combinations of techniques. For instance, hardening a single component to achieve high reliability might be extremely costly and inefficient compared to a system-level approach or a combination of both. Figure 1.7 illustrates the concept with two systems, one with a hardened instance of a component and another with two instances of the unhardened component, which can also be a sub-system.

An example of a cross-layer approach is seen in the ASTEROID project [10, 38], in the frame of which part of the work reported in this thesis was developed. The project was a member of *Deutsche Forschungsgemeinschaft* – German Research Foundation (DFG)'s Priority Program SPP 1500 that focused on cross-layer approaches for dependable embedded systems [63, 64]. ASTEROID developed a cross-layer fault-tolerance solution for mixed-criticality that increases the system's reliability at a higher level of abstraction without resorting to hardware redundancy [10, 38]. Aiming at efficiency and off-the-shelf hardware, ASTEROID combined error detection features in hardware and software with error handling and recovery in software. More details of the overall approach of ASTEROID is given in Chapter 6.

Another concept that is highly relevant in this thesis and in the context of cross-layer approaches is *resilience*. Resilience can be seen as a form of fault tolerance [7] and is defined here as the capability to recover from errors and to resume correct service, tolerating possible data loss. Resilience is relevant in cross-layer approaches as it allows the use of redundancy in time in higher layers of abstraction – e.g., packet retransmission or software re-execution. As seen later in Chapter 2, the lack of resilience prevents the

use of redundancy in time due to the unavailability of the lower layers due to errors. Notice, that resilience *per se* does not ensure the integrity of the system, which should be achieved by other means.

When designing safety-critical components, sub-systems and systems, high dependability is recurringly the requirement. Nonetheless, sooner or later, depending on the failure rate, the component or system inevitably fails. That raises yet another important feature, which dictates the behavior upon failure:

- Fail-safe behavior; or
- Fail-operational behavior.

In the former, the component or system enters a safe state upon service interruption. An example thereof is an Electronic Control Unit (ECU) connected to a Controller Area Network (CAN) bus with a CAN transceiver. When the ECU fails, it fails silently, not flooding the bus and affecting the performance of the remaining ECUs [98]. In the latter, the component or system is fault tolerant and continues operating albeit with reduced functionality until transitioning to a safe state. Alternatively, ISO 26262 defines the fail-operational as an *emergency operation* mode, which can be triggered by a *safety mechanism*, with possible service *degradation* (reduced functionality, performance or both) for providing *safety* until a transition to a *safe state* is possible [72]. An example of the latter is an ADAS system, which must continue (at least minimally) operating in spite of a failure until the car has been safely stopped or taken over by a human driver [98]. Thus, the requirement depends on the intended functionality of the component in the context of the system – i.e., how it integrates into the system and interfaces other components.

1.5.1. Fault-tolerant system architecture

Hierarchical system architectures can be used to achieve fault tolerance and fail-operational behavior. In fact, hierarchical designs are defined by ISO 26262 as a *highly recommendable* principle for highly critical systems (ASIL C and D) and *recommendable* for the lower critical ones (ASIL A and B) [72]. Such an architecture comprises two or more subsystems, which may be identical or not. When identical, it becomes the well-known n-modular redundant architecture, requiring k-out-of-n subsystems to survive in order for the system to survive ($n \geq k$) – i.e., the system fails when $n - k + 1$

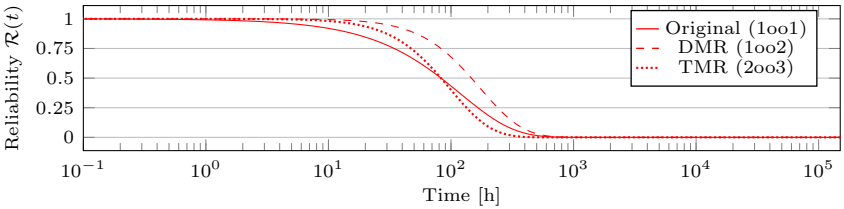


Figure 1.8.: Reliability of classical redundant systems.

subsystems fail [67]. It inherits therefore its properties: MTTFs of $1/\lambda \cdot \sum_{x=k}^{n} 1/x$ at the cost of *at least* n times the original system [67].

To illustrate what can be achieved with classical n-modular redundancy, Figure 1.8 plots the reliability in time of classical redundant systems. It considers the system from Figure 1.6 as the original one (1-out-of-1) and it considers ideal voters. Although, a 2-out-of-3 system, a classical Triple Modular Redundancy (TMR) configuration, can extend the high reliability up to a certain point in the short term, in the long term it actually results in a system that is less reliable than the original one: the MTTF is 5/6 that of the original one. A 1-out-of-2 system can extend the reliability in time but only in a Dual Modular Redundancy (DMR) configuration that is capable of detecting which redundant system failed – i.e. pin-point the failure. Usually, DMR can be used to detect a failure but cannot pin-point it, whereas TMR can detect up to two failures and can pin-point the first failure. An efficient and attractive alternative is to employ non-identical subsystems, whose common functionality must be only the critical one in order to provide the fail-operational behavior. Not only are unnecessary costs reduced therewith, but also common cause failures are avoided.

An example of a hierarchical fault-tolerant architecture that is capable of fail-operational behavior is illustrated in Figure 1.9 where a system is composed of a high-performance layer, a fall-back layer and a voter. ISO 26262 also to such architecture as dynamic redundant architecture with standby [72]. The high-performance subsystem implements the critical functionality and additional desirable functionalities. The second subsystem, the fall-back, implements only the critical functionality. The critical functionality is thus ensured in case either subsystem fails. The system provisions for a single failure scenario, where only one of the components fails. That is, either one of the subsystems fail or the voter fails. Upon failure of

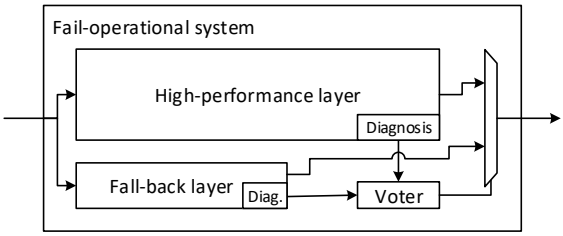


Figure 1.9.: Example of a fail-operational system with two subsystems: high-performance and fall-back.

the high-performance layer, the transition to the fall-back layer is called *fail-over*, which is performed before any faulty output leaves the system in order to maintain integrity. Notice that the voter may also fail. Therefore, the voter must fail safe – i.e., it must chose either layer as output, ensuring the operation under the single failure scenario.

In the occurrence of a failure, the system must then remain operational until reaching a safe state [72]. For instance, in case of a motorized vehicle, a plausible requirement is that the system must remain operational until the vehicle has safely stopped. Given the first failure, the probability of a second failure until the system reaches a safe state is usually acceptable. The probability of a second failure, and thus the probability that the entire system fails, is thus usually accounted as *acceptable risk* in ISO 26262 [72]. How much risk is acceptable, in turn, depends on the final application – e.g., a car would require much less time to reach a safe state than an airplane in cruising altitude that would require one hour to land.

1.5.2. Error detection vs. error recovery

In order to satisfy the strict integrity requirements of high criticalities [72, 35], all errors and deviations from the correct behavior must be *detected* and contained before a wrong output propagates outside the system. To meet the dependability requirements, the system's fail-operational behavior continues providing the critical functionality. Finally, the system must operate in a timely manner in order to meet the timing requirements.

The next goal after ensuring the critical functionality is to provide improved quality and experience to the user with extra robustness. Fig-

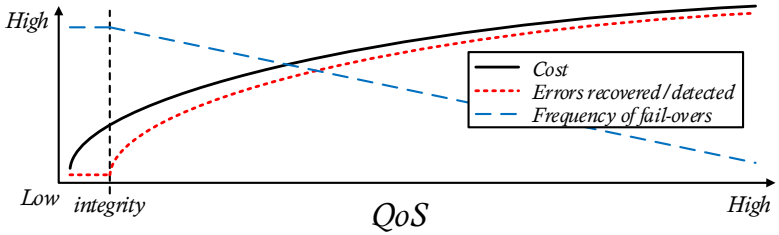


Figure 1.10.: The quality vs. cost vs. fail-overs trade-off.

ure 1.10 illustrates the cost of achieving integrity and the cost of further increasing the dependability of the system. By *recovering* from more errors (dotted curve), the frequency of fail-overs can be reduced (dashed curve), therewith increasing the quality of the service. The perceived quality of the product increases therewith.

An example is the aforementioned cross-layer ASTEROID approach. It relies on error detection mechanisms in hardware for efficiency and lower execution time overhead and delegates the task of handling errors to software [10, 38, 127]. The approach then supports different strategies to handle detected errors, which can be applied based on the requirements of functions implemented by the system. For example, best effort tasks can simply be restarted, whereas highly critical tasks can be recovered, e.g., by means of checkpointing mechanisms [127].

Notice that, in a hierarchical fault-tolerant system, errors that are detected and handled in a certain layer (or level of abstraction) are seen as masked in the layers above. When errors are only detected and not handled in a certain layer, the unhandled error is then reported to the layer above, which will then either handle it or report it further. For instance, DMR can detect that the service of the two sub-systems differ and it can thus report to the system so that it can take appropriate action. Also, in cross-layer approaches, some errors can be handled locally and the remaining, unhandled errors can be reported and handled in the layers above. Thus, a *failure* is then an error that cannot be handled in the highest layers of the system. Notice that, by definition, an *undetected error* affecting the service of the system is a safety violation and considered an unsafe failure [72]. The choice of which errors should be detected, which should be handled and where that should be done has, naturally, a direct impact on cost

and QoS.

Thus, a clear distinction can be made between detecting errors and recovering from them. Error detection is crucial. Integrity cannot be achieved without it. On the other hand, the strategy with which the error is handled can be chosen according with the system requirements, the available budget or the available technology. Thus, higher product quality requires tolerating more errors in the high-performance layer, whereas the budget product can be safe and ensure the minimal functionality while potentially suffering more disruptions of the additional services and high-quality functionality.

1.6. Research objective and contributions

As introduced throughout this chapter, dependability has many facets that must be addressed when designing a system. That is specially challenging in real-time mixed-critical systems, where safety standards play an important role and where responding in time can be as important as responding correctly or even responding at all. This thesis addresses the issue of cross-layer dependability in many-cores for real-time mixed-critical systems. It addresses the question of what is the impact of random hardware errors on a real-time many-core system, with special focus on the NoC, and what must be done to overcome such errors and achieve a reliable service with timing and integrity guarantees.

The *grand objectives* of this work are ① to ensure that any errors in are detected before propagating throughout the system, thereby ensuring its timing and integrity, and possibly recover from errors, with focus on the NoC; and ② to efficiently handle errors in a cross-layer solution with replicated software execution.

The *minor objectives* of this work are: to formally analyze the timing of the proposed techniques; to evaluate the proposed techniques with respect to their hardware cost and performance, where applicable; to compare them with the state of the art; to keep the overhead required to achieve high dependability minimal.

The core **contributions** of this thesis are:

- Uncovering and description of the impact of random hardware errors on real-time NoCs and on the system (Chapter 2);

- A resilient, real-time mixed-critical NoC architecture (Chapter 3);
- Proposal of Automatic Repeat reQuest (ARQ)-based transport protocols for real-time NoCs and their efficient use (Chapter 4);
- Proposal of AIQ, a lightweight mechanism to guarantee integrity and predictability in mixed-critical, real-time NoCs (Chapter 5);
- Proposal of the replica-aware co-scheduling for mixed-criticality (Chapter 6); and
- The formal timing analysis of the proposed approaches.

The core contributions of this thesis have been previously published in conference proceedings and scientific journals. The contents of those publications have been used in this thesis in the form of text, figures and data. A complete list of publications that are related to this thesis, including brief descriptions, is given in Appendix B.

1.7. Overview

This thesis is organized as follows. The goals and challenges motivating the work on dependable mixed-critical real-time systems are introduced in Chapter 1. Chapter 2 reports the impacts of random hardware errors starting at the NoCs-level and ending at the system-level. Based on those findings, a set of mechanisms to attain dependability is introduced in the sequel. Chapter 3 introduces the resilient NoC architecture for mixed-criticality, which is able to provide a highly reliable soft error protection that is transparent to software. Data can be reliable transported over the proposed resilient NoC by means of ARQ-based protocols. Chapter 4 proposed the efficient use of transport protocol in NoCs under real-time guarantees. Aiming at a low-overhead NoC with guaranteed integrity and timing under errors, the AIQ approach is proposed in Chapter 5. AIQ ensures integrity and real-time guarantees without recovering from errors, which can be handled in software in a cross-layer approach. The handling of errors in higher layers of cross-layer approaches is addressed with replicated software execution. Chapter 6 proposes the replica-aware co-scheduling for efficient resource utilization in mixed-critical many-cores. Finally, Chapter 7 concludes with remarks and an outlook.

2. The Impact of Soft Errors

Random hardware errors can be classified in two types: hard errors and soft errors [61]. Soft errors in hardware are events where data is corrupted and the device is not permanently damaged, whereas in hard errors the device fails permanently. This work focuses on the former, which is orders of magnitude more frequent. Soft error is a term commonly used to refer to a subset of Single-event effects (SEEs) [61]. SEEs are induced by the interaction of primary or secondary ionizing particles with electronic components [52]. Primary particles are heavy ions in space or alpha particles. Secondary particles, or recoils, are the result of the nuclear interaction of particles, such as neutrons or protons, with atoms of the die [52]. According to Heijmen, SEEs can be classified into the following categories:

- Single-bit upset (SBU): a bit-flip (upset) caused by a particle strike;
- Multiple-cell upset (MCU): the upset of two or more memory cells or latches;
- Multiple-bit upset (MBU): the upset of two or more bits in the same word;
- Single-event transient (SET): a voltage glitch in a circuit, which only becomes an error when captured by register;
- Single-event functional interrupt (SEFI): loss of functionality due to the perturbation of, e.g., control registers and clock and reset signals;
- Single-event latchup (SEL): the event creates a high-current state by triggering a parasitic dual bipolar circuit, requiring a power reset.

This chapter focuses on the effects of SBUs, MCUs, MBUs and SETs. SEFIs and SELs are not considered and are thereby assumed to be either handled at the device or circuit-levels, or their occurrence is considered as a failure if,unhandled, their effects differ from the other types.

Ebrahimi et al. showed that errors rates are decreasing with the technology scaling, which contradicts early predictions [20] that have not been confirmed thanks to improvements in EDA tools and in the device. In-

terestingly, Ebrahimi et al. also showed that the occurrence probability of SETs increases with the frequency whereas the occurrence probability of SBUs decreases with the frequency [41]. Although SEEs have the same cause, their impact on the system depends on when and where they occur.

This chapter is organized as follows. The current state-of-the-art in the analysis of errors in NoCs is reported in Section 2.1. Section 2.2 summarizes the analysis methodology employed in this work, including an extension to classify error effects. The analysis methodology is then applied to the IDA NoC and the results are described in Section 2.3. Section 2.4 discusses the impacts of errors in the system level. Finally, Section 2.5 summarizes the chapter.

2.1. Related work

Fault-tolerance for NoCs has been extensively researched in the last 10 years [121]. However, research has addressed only specific errors and, consequently, failure modes. In the context of mixed-critical systems, where strict requirements must be met in certification processes [72, 35, 69], addressing errors individually is not sufficient. In a mixed-critical system, all errors and their impacts must be taken into account, requiring a holistic analysis of the NoC. A proof in certification processes usually involves an FMEA [68].

The FMEA is a comprehensive bottom-up approach that systematically captures all failure modes and their effects on the system [67, 68]. For each component, failure of components in the lower levels of a system and then evaluates its effects and their propagation throughout the system. Since they propagate, the effects can be observed in different interfaces, e.g., locally as well as globally. The FMEA can be extended, for instance, by assigning criticalities or priorities to effects. That extended version is known as the FMECA [67, 68].

Radetzki et al. present in [121] an overview of the state-of-the-art in fault-tolerance of NoCs. They summarized and reviewed failure mechanisms, faults models, diagnose techniques and fault-tolerance methods. Additionally, they present the application of the methods in the three layers of the network stack: data link, network and transport layers. Despite the large number of mechanisms developed in previous years, they point out the need for holistic, goal oriented approaches that are better linked to

the physical causes. Marculescu et al. give an overview about outstanding research problems in NoC design and also list fault-tolerance and reliability as key problems. The motivations given in [95, 121] and many related works are however the increased transient faults caused by transistor scaling and not the certification for safety-critical systems.

The impact of circuit-level timing violations caused by process and temperature variations was researched by Aisopos et al. in [2]. They proposed a system-level error model where the functional effect of faults are modeled, such as flit derouting, loss and duplication. However, the model is intended to detect only circuit-level timing violations and does not cover SEEs.

SoC-level approaches have also been researched. An FMEA methodology for SoC-level design compliant with IEC 61508 [69] was proposed in [96]. The methodology extracts sensible zones from the Register-Transfer Level (RTL) hardware description and use them as the basis for an FMEA. A sensible zone is an elementary failure point of the SoC in which one or more faults converge to lead to a failure [96]. They can be memory elements (e.g., registers), primary inputs and outputs, and critical nets (e.g., clocks). The proposed methodology was applied to design memory subsystems for microcontrollers. Another work presents a SoC-level risk assessment using a SystemC Transaction-level Modeling (TLM) model [27]. Although it seems sufficient for risk assessment, it does not yield sufficient insight into error propagation.

Numerous studies have addressed transient hardware faults [140, 39, 118, 81, 19], whereas others have put effort into comparing different fault-tolerant schemes. Murali et al. studied in [104] the reliability-energy and the performance-energy trade-offs for different error correction schemes: ARQ and Hybrid ARQ, where ARQ is combined with Forward Error Correction (FEC). The authors of [43] and [116] concentrated on the triple performance-reliability-energy consumption trade-off, the latter focusing on router architectures.

Permanent hardware faults were investigated in [50], which proposes a highly resilient NoC architecture. The redundancy in the network and in the routers (e.g., buffers and decoders) is used as means to provide robustness though reconfiguration without resorting to N-modular redundancy-based solutions. Later on, Tsai et al. present a fault-tolerant scheme exploring bidirectional channels between routers in [151]. Instead of deflecting

packets away from faulty channels, as in traditional schemes, a bidirectional channel is used in half-duplex mode when faulty channels are present.

For off-Chip networks, there is an even longer history of studies on reliability including FMEA, as seen in [141]. While some ideas are applicable to on-Chip networks, that is not always the case since in NoCs there are limiting factors, such as area, power and timing constraints. Besides, the links are the dominating factor in the reliability of off-Chip networks, whereas the routers dominate in on-Chip ones. In off-Chip networks, the objective of FMEA studies is usually to increase the availability of the network. This work, on the other hand, aims at creating the base for a resilient NoC with minimal overhead and which is compliant with safety standards.

Although extensive research has been carried out on fault tolerance for NoCs, only certain mechanisms are presented and evaluated. Therefore, one can not prove that all SEEs are covered, as required by safety standards, since a systematic assessment of all potential failure modes is required. To fill this gap, Ahrendts derived in [1] a methodology based on the FMEA to evaluate the impact of SEEs on NoCs.

2.2. Failure Mode and Effects Analysis (FMEA)

In the FMEA, each instance of each component type is analyzed [68]. Each possible failure mode of the component instance is then examined for every possible system state, evaluating local and global effects of the failure mode. A failure mode is the specific way in which a failure occurs in terms of failure of the component function under investigation. Local effects concern how the functionalities of individual sub-components or the local switch are affected. Global effects concern how the functionality of the NoC as a system is affected, i.e. error propagation.

If applied directly, the FMEA leads to an explosion of cases to be evaluated. Since all combinations of components types, their instances and the states of each instance must be evaluated, the number of cases increase prohibitively for a real system, exceeding reasonable limits. To avoid that, techniques can be applied to reduce the analysis complexity.

2.2.1. FMEA-based analysis methodology for NoCs

When analyzing the impact of soft errors on NoCs, Ahrendts derived an FMEA-based methodology for NoCs that keeps the analysis complexity within reasonable limits [1]. The analysis is performed on the block-level and assumes errors occur on all connections between blocks and between adjacent switches. The errors are evaluated individually for each logical signal, such as the actual flit data or the synchronization and control between modules. In the methodology, three techniques exploit properties of the NoC to reduce the analysis complexity: error abstraction, symmetry exploitation and worst-case effect on a test packet. Those are summarized next.

Error abstraction: Instead of evaluating single-bit errors individually, signals are grouped logically by their function, dramatically decreasing the number of cases to be evaluated. By assuming that every single-bit error has an immediate effect on each output signal of the component, the results are still conservative after the simplification. Errors are considered equivalent if their effect on a flit transmission is identical (e.g., all errors in the packet's payload are considered equivalent). Furthermore, only errors at the block-level are considered, assuming that errors occurring within a block will either eventually show up at its interface or are masked and hence irrelevant – i.e., latent errors are irrelevant if they never show up at the interface. The internal state of the blocks are considered when determining the effects of each error.

Symmetry exploitation: Each switch has multiple inputs and outputs that behave identically. Hence, it is not necessary to analyze all combinations of input and output paths of every switch separately, the analysis of one such path is sufficient. Likewise, a mesh network shows similar symmetry between switches for different paths through the network. Thus, one does not need to consider a large network, but can construct a minimal network configuration which still shows all effects of possible path segments of larger networks.

Worst-case effect on a test packet: A single error in a switch may have very different consequences depending on the current state of that switch. The state is defined by the packets being transferred and the respective state of those transfers. To avoid considering all possible combinations of transfers and their states, which would result in a tremendous state space, only the transmission of one test packet – using different flit

compositions – along a selected path is considered (which is sufficient due to symmetries). For that packet, all possible error effects on the packet itself and on any potential background load are evaluated. For background transfers, only the worst-case transfers and states are considered – i.e., those that actually interact with the test packet. Since the background traffic and the test traffic are symmetric, uncovered errors affecting the test traffic do not need to be evaluated again for the background load.

The methodology draws the system boundary between the NoC switches and the network interfaces that are used to connect the IP blocks (e.g., processors and memory). That decision aims at modularity, since different network interfaces may exist in the system depending on the type of IP they are connected to. Regarding the network interfaces, the packets are assumed to be injected fault-free in the NoC. In case they are injected faulty, the errors are a subset of the ones found in a packet transmission between switches. Moreover, the analysis includes the state machine that handles flits received from the link, in order to reveal the effects of an error in a switch (up to the network layer). This does not affect modularity since this state machine does not change between different NIs in the same NoC. They only depend on the packet formats supported by the network and are essentially the same as the state machine in the switch.

2.2.2. Qualitative and quantitative extension

The methodology for NoC analysis was then extended to qualitatively and quantitatively summarize the output of the FMEA [128, 129]. The output of an FMEA is an extensive spreadsheet output of the data report, e.g., as seen in [124]. Thus, a qualitative and quantitative summary of the results of the FMEA enables a faster understanding of the vulnerabilities in a NoC design and a faster response by the designer. The extension starts with the characterization of the effects followed by a probability assessment.

2.2.2.1. Effects characterization

The error effects are classified by their duration on the NoC, based on the error classification introduced in [28]. At first, two classes of effects were identified, the conventional ones: *transient* and *static* effects [129]. Later on, with modifications in the NoC trying to prevent, other classes of effects were observed, such as *degrading* and *intermittent* [129].

- **Transient:** the error vanishes with the affected packet – e.g., packet payload corruption;
- **Degrading:** the error degrades performance. It accumulates due to new occurrences and eventually manifests itself – e.g., credit counter fails does not update and credits are erroneously lower;
- **Intermittent:** the error remains in the switch but can eventually vanish – e.g., a VC was not released by a corrupt packet and will only be released by a subsequent healthy packet traversing the switch in the same direction.
- **Static:** the error remains in the switch affecting subsequent transmissions – e.g., the VC reservation is never released.

Since we are analyzing a NoC for mixed-critical systems, we also want to understand how task isolation can be violated. Task isolation means that the misbehavior of a task will not impair other tasks in the system, also called *sufficient independence* [72]. Thus studying the effects that a faulty transmission of one task has on other tasks in the NoC is crucial in order to prevent them properly. That resulted in another class of errors called *isolation violation*.

- **Isolation violation:** the error, regardless of its duration in the NoC, affects the transmission of other streams – e.g., by corrupting or causing a packet to be lost.

This is not an exhaustive list of classes, as there may be specific cases in a particular architecture. Nonetheless, it serves as a starting point and guideline for the more general cases.

2.2.2.2. Effects probabilities

After classifying the error effects, the designer already has a good insight into the weaknesses of the NoC's architecture when exposed to faults. The next step is to assess their probability of occurrence of the failure modes. This allows the designer to differentiate errors that are very unlikely and the others very likely to occur.

The above-described analysis methodology conservatively considers that any fault in a block leads to an error. As a result, the occurrence probability of an error on an output signal of a component, i.e. a failure mode, can be calculated as the probability that a fault happens in that component. This

extension aims at assessing relative occurrence probabilities without being bounded to a specific technology node and without going into details of the underlying technology. Thus, the relative probability assessment is based on the *area* consumed a component on the chip. That can be extracted or estimated from metrics provided by, e.g., synthesis reports of Xilinx Vivado [157]. Deriving accurate, absolute error occurrence probabilities falls outside the scope of this work.

For Field-Programmable Gate Arrays (FPGAs), one can assess probabilities relative to the number of registers used by each component in the switch. Alternatively, one can assess probabilities relative to the number of Look-up Tables (LUTs) used by each component. According to [129], the difference between using LUTs and registers is smaller than 0.21%, thus throughout this chapter, we report results with registers. For Application-Specific Integrated Circuits (ASICs), one can assess probabilities relative to the area occupied by the cells of the design, as reported by, e.g., Synopsys Design Compiler [145].

The assessed probability accounts for the direct effects of an error. The indirect effects are not considered – i.e., the probability that one switch is affected by effects propagated from errors in a different switch.

2.3. Impact of soft errors on NoCs

This section describes the impact of soft errors on an unprotected version of the IDA NoC, introduced in Section 1.2. The FMEA analysis of the IDA NoC uncovered 107 failure modes within the switch and 54 failure modes in the link between switches. The interested reader can find the complete output of the FMEA in [124]. Next, the failure modes are summarized with respect to their local effects on the switch and their global effects on the NoC.

2.3.1. Local and global effects

Each failure mode can present one or more local effects. An example is that a flit can be forwarded to the wrong output port. In that case, the original flit that was supposed to be forwarded to an expected port is lost and an unexpected flit is forwarded to a different port.

The local effects of a soft error were found to be of 6 different types:

- **Error masked:** The subsequent block ignores it. It occurs when the faulty signal is only evaluated when other signals have a specific value;
- **Flit corruption:** The flit content changes. Depending on where this error occurs inside the flit, there may be different global consequences;
- **Flit loss:** It may happen when an error affects the communication between blocks – e.g., the input module sends a flit to the switch fabric but it doesn't get through, or the corruption of a flit's VC field causing the flit to be stored in an incorrect VC buffer;
- **Flit sent to wrong output port:** It may be caused by an error affecting the control information – e.g., input port select, or an error in the communication between blocks – e.g., between arbiter and switch fabric;
- **Flit transmission delayed:** It may be caused by an error affecting the control information – e.g., the VC priority, or an error affecting the communication between blocks – e.g., register bank and switch arbiter. Arbitration errors usually do not lead to a complete loss of functionality since the switch re-arbitrates every cycle;
- **VC buffer blockage:** It may occur when an error induces an incorrect decision regarding VC reservation in a switch. Such allocation error is usually permanent since a reserved VC is only released when the tail of a packet is processed, which will never happen as the reservation error prevents progress.

Globally, a soft error can cause 5 different effects, observable at the NoC-level. The global effects depend on the type and location of the errors. One error can present one or more global effect. For instance, an error can cause the corruption of a packet and at the same time cause the loss of a packet of another traffic stream.

The global effects of a soft error in the NoC were found to be:

1. **QoS violation:** The VC QoS guarantee is violated. This may happen temporarily – e.g., due to an incorrect switch decision or a signal glitch, or permanently – e.g., due to corruption of the VC priority or due to VC buffer blockage;
2. **Packet loss:** The packet is lost in the network. The packet might also be delivered to a wrong recipient. This may be caused by the

loss of the head or tail of a packet (HF or TF), corruption of the route or VC fields, an incorrect routing decision in a switch, or VC buffer blockage;

3. **Packet corruption:** The packet arrives at the correct destination but its content is corrupt. This may happen, e.g., due to a bit flip in the payload of a flit or the loss of a BF;
4. **Return route corruption:** The route field gets corrupted without immediate effect – e.g., because the error affected only the part of the route that was already traversed, causing the Network Interface (NI) to be unable to reconstruct the return route for a response or acknowledgement;
5. **VC buffer blockage:** This may be caused by the loss of the head or tail of a packet (HF or TF) or by the incorrect switch decision caused by corrupt control data – e.g., credit counter and VC access control. This effect appears often together with packet loss, although they may happen independently. Due to wormhole switching, the error may propagate to the downstream switches – e.g., when a packet's tail (TF) is lost causing all switches downstream to never release the corresponding VC reservation due to the missing TF.

The effects of the corruption of the flit type and VC fields can also be considered as a flit loss. The corruption alters the flit semantics and, in the perspective of its aggregating packet, the flit is lost. Nonetheless, the flit still exists in the NoC and will affect the transmission of other packets.

2.3.2. Effects characterization and probabilities

The effects were characterized with respect to their duration and isolation violation, as described in Section 2.2.2.1. The metric used here to show the results is the percentage of cases where a global effect (1–5) presents a given characteristic. As a reminder, an effect is *transient* when the effect goes with the affected packet and it is *static* when the effect remains in the NoC affecting subsequent transmissions.

The percentage of cases where a global effect presents a given characteristic is reported in Figure 2.1. The figure shows the effect duration on the left-hand side. Global effect 1 usually presents transient effect duration, whereas effects 3 and 4 are always transient. Global effects 2 and 5 are critical. Effect 2 remains in the NoC in more than 57% of the cases,

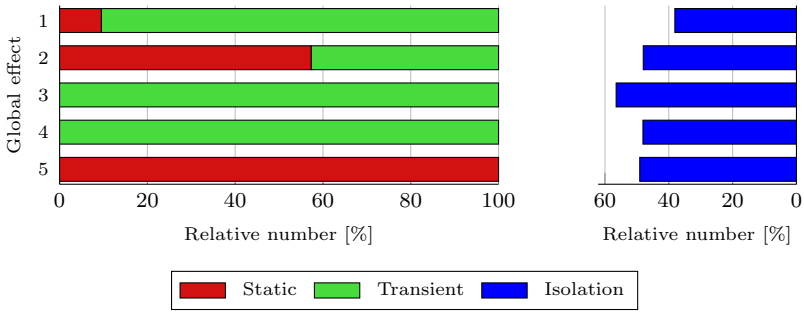


Figure 2.1.: Global effects characteristics: relative number.

whereas effect 5 remains in all cases. Global effects 2 and 5 are caused by the corruption the state of the switch by the error. The state of the switch is defined by the packets being transferred and the state of those transfers. In the occurrence of the local effects flit corruption (VC field), flit loss, flit sent to wrong output port and VC buffer blockage, the state of one or more transfers don't reflect the reality anymore.

The right-hand side of Figure 2.1 shows the relative number of cases where a failure mode violates the isolation property. Notice that this characteristic is independent from duration – i.e., transient or static. The isolation is violated when a flit deviates from its route and affects other transmissions in the NoC either by migrating to another VC or by changing its route. The local effects flit corruption (VC and route fields) and flit sent to wrong output port are responsible for the violation.

Nonetheless, different errors have different occurrence probabilities and so do their effects. To make evident which effects are more critical, the relative probability of a global effect was assessed, as described in Section 2.2.2.2. The numbers were obtained using synthesis results for a Xilinx Virtex-6 FPGA [128, 129].

Figure 2.2 shows the relative occurrence probabilities of the global effects. Among all effects, effects 2, 3 and 5 are the most likely to happen. That is due to the impact of the input buffer on the occurrence probability of an error that causes flit corruption or loss. The input buffer stores approximately 85% of all data in the switch. Effect 1 presents almost exclusively transient duration (very small probability of a static effect), whereas effect 4 presents only transient duration. Both less likely to occur

than the others.

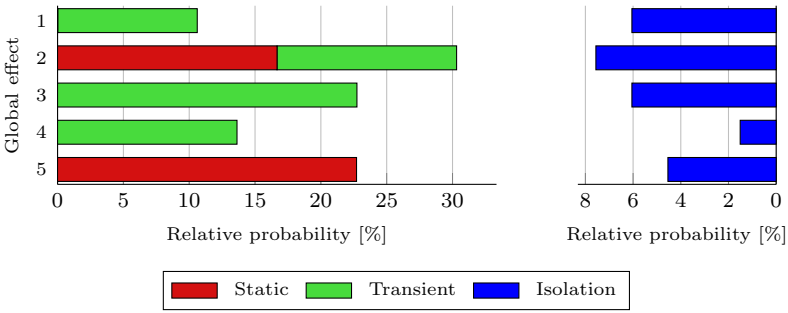


Figure 2.2.: Relative occurrence probabilities of global effects.

2.3.3. Assessment of uncovered effects

The analysis results revealed the nasty effects that a soft error can cause in the NoC. As is, soft errors cause not only transient effects but also static effects. The occurrence of a static effect causes partial unavailability of the NoC and renders end-to-end mechanisms, such as ARQ, useless. The static effect is similar to one of a hard error caused by a permanent fault, but different in that the effect goes away when resetting the circuit. The biggest problem with such static effects caused by soft errors is that they have a much higher occurrence probability than the ones caused by hard errors (cf. Section 1.4). Thus, handling them as hard errors would either require very expensive overprovisioning or result in a low quality service, as discussed in Section 1.5.1.

The major source of static effects was identified as the inability of the switch to handle unexpected flits – i.e., inability to handle unexpected input. For instance, when handling head-less and tail-less packets. The VC flow-control does not allow a packet to be transmitted in a reserved VC in a given output port when that VC is reserved for another packet. In case of a tail-less packet, the respective VC reservation will not be released and other packets will be blocked indefinitely in the input buffer. The other case is the head-less packet, where the BF or TF is the first flit of the packet to arrive at the IB. Since the received flit does not contain routing data and a VC reservation does not exist, the flit will remain in the input buffer.

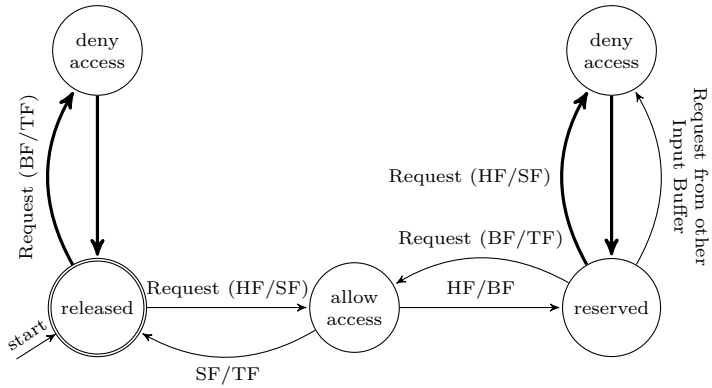


Figure 2.3.: The VC flow-control state machine.

The state machine implementing the VC flow-control is shown in Figure 2.3. The state machine controls the second stage of the switch arbitration, where an input buffer requests access to a VC in a given output port. The first arbitration stage controls the access to an output port. In the figure, all the requests in the conditional transitions belong to the same input buffer, except for the rightmost transition from *reserved* to *deny access*. The first case happens when the VC is *released* and an input buffer requests access to transmit a BF or TF (bold cycle on the left-hand side). The second case happens when the VC is *reserved* and the input buffer requests permission to transmit a HF or SF (bold cycle on the right-hand side). In both cases the access will always be denied, effectively blocking the VC.

2.3.4. Early exploration of analysis-enabled resilience improvement

In an initial effort to transform the static effects into non-static ones and, thus achieve resilience, the state machine of the VC flow-control was altered in two ways:

- Drop flits that cannot be forwarded – i.e., a head-less packet;
- Transfer the VC reservation to the next packet entering the switch through the same input port as the packet that has the VC reserva-

tion when the switch detects it was not properly released – i.e., when an input port receives a packet after a tail-less packet.

The changes are depicted in Figure 2.4, where the transitions represented by the dashed edges were removed and the dotted ones were added. Now, in the first case, when a VC is *released* and a head-less packet arrives, the access is denied and the flits will be eventually dropped, allowing the next packet to be served. In the second case, when a VC is *reserved* – e.g., not correctly released due to a tail-less packet – and a new packet arrives at the same input port, the new packet takes over the reservation. The case where the VC is *reserved* and a different input buffer sends a request remains unchanged: the access is always denied, since no safe assumption can be made. The changes are located in the link layer of the NoC and are thus also applied to the NI.

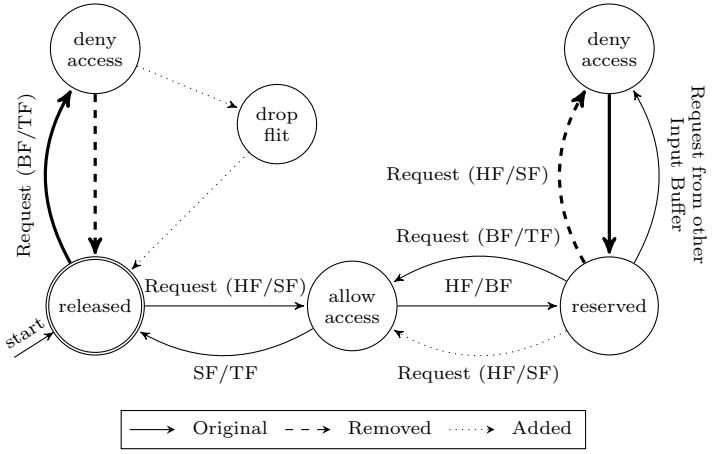


Figure 2.4.: The improved VC flow-control state machine.

The improved VC flow-control helped prevent cases of errors with static effects. First, let us consider the percentage of cases where a global effect presents a given characteristic. Figure 2.5 plots the numbers for the *improved* version of the NoC (I). To facilitate the comparison, the figure also plots the numbers for the baseline (B) version from Figure 2.1. The changes resulted in a visible improvement by ruling out static effects. However, two new classes of effect emerged: degrading and intermittent – both less compromising than static. A *degrading* effect degrades the per-

formance, accumulates due to new occurrences and then manifests itself – e.g., error reduces the utilizable input buffer space until it violates the QoS guarantees. The *intermittent* effect means that the effect can go away, as opposed to static, but it takes longer or is less likely to happen than a transient one – e.g., requiring a second packet to traverse the switch in a specific direction. With respect to *isolation*, the number of occurrences decreased in all cases, 14.77 percentage points in total.

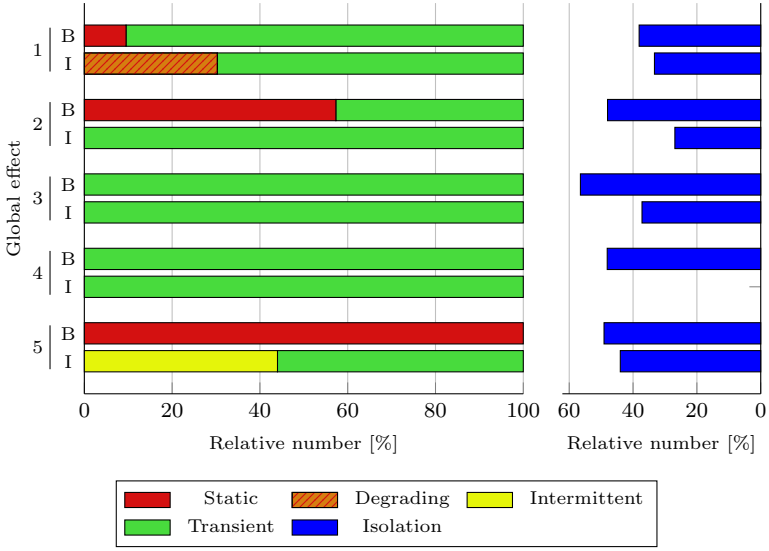


Figure 2.5.: Global effects characteristics: relative number.

The occurrence probabilities of the global effects are reported in Figure 2.6. For *improved* (I), the reduction of the probabilities of 2, 4 and 5 increased the ones for 1 and 3. This can be explained by the fact that, since static VC blocking (5) does not occur anymore, fewer packets are lost due to corruption (2) and, as a result, those packets are delivered despite being corrupted (3). They also may corrupt other packets and consequently isolation violation increases. Return route corruption (4) decreased due to the improved VC flow-control at the NI. Finally, QoS (1) increased for two reasons: the effect is not being masked by VC blocking anymore; and because the values are relative – 1 and 3 increase to compensate the decrease in 2, 4 and 5.

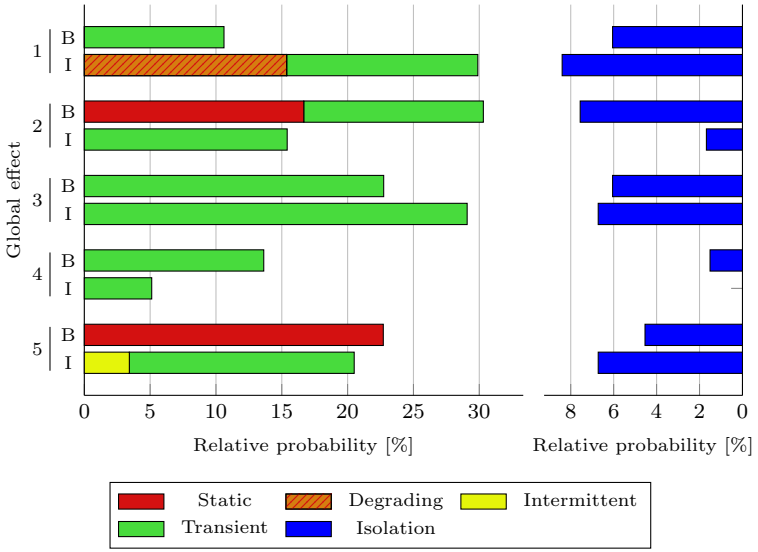


Figure 2.6.: Relative occurrence probabilities of global effects.

Enabled by the FMEA, a small change to the design resulting in the improved VC flow-control allowed the static effects to be ruled out. However, non-transient effects still remain in the NoC, impairing its resilience and applicability in safety-critical real-time systems. Although the VC flow-control was a main source of static effects, achieving resilience and dependability in the NoC requires several additional measures. In Chapter 3, those issues are addressed and a resilient NoC for safety-critical real-time systems is introduced, including a resilient VC flow-control, which extends the VC flow-control presented in this chapter.

2.4. Impact of soft errors on the system

The impacts visible at the software level have been summarized by Rebaudengo et al. in [134]. According to Rebaudengo et al., soft errors can affect the software in two ways: it can affect the *data* or the *execution flow*. The classification can also consider whether the error affected an application or the operating system [37, 136]. Figure 2.7 summarizes the error classifica-

tion. This section addresses the impact of errors affecting the application and the operating system separately and summarizes the impacts at the system-level.

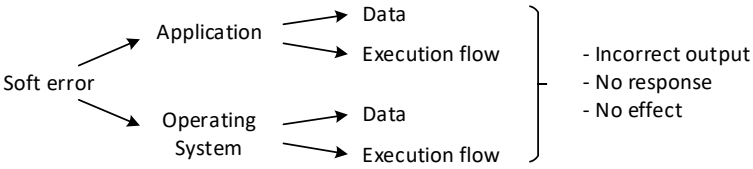


Figure 2.7.: Overview of impact of soft errors in software.

The impact of soft errors on a real-time operating system has been recently evaluated with irradiation experiments [136]. The software consists of dOSEK, a dependability-oriented static embedded kernel [66], and a generated application with 125 pairs of threads that communicate, where each pair of thread communicate with each other. A Xilinx Zynq^{TM}-7000 AP SoC in 28nm Complementary Metal-Oxide-Semiconductor (CMOS) with two ARM® Cortex^{TM}-A9 cores with the maximum frequency of 667MHz was employed. In the setup, only a single core was used and all hardware-based soft error protection mechanism was disabled and only the SoC was radiated, thus the rest of the platform, including the external Dynamic Random-Access Memory (DRAM) chips, was error free. The SoC was submitted to a neutron flux of approximately $10^6 n/cm^2 s$ with energies above $10 MeV$. The results showed that it is more likely for the system to hang – i.e., to freeze without producing output – than to fail producing incorrect output. However, as pointed out by the authors, the outcome of an error is highly dependent on the application and the focus of the evaluation was the kernel.

The impact of soft errors in the application has been investigated by [37, 36, 144]. Döbel et al. performed, in the context of the project ASTEROID, fault injection campaigns targeting applications of the MiBench benchmark suite [56] executing as user-level applications. Faults were injected in the functional units of the processor, affecting the instruction decoder, arithmetic logic unit, register bank and register allocation table. The outcomes of the simulations were classified in four categories:

- *success*, where the execution finished and produced the correct result – e.g., an error affects a variable that is overwritten before being read,

masking the error;

- *crash*, where the execution was terminated prematurely with an abnormal result, also known as a fail-crash – e.g., an error causes the program to access an illegal memory region and to be terminated by the Operating System (OS);
- *infinite*, where the execution did not terminate within a specified amount of time – e.g., an error caused the execution to enter an infinite loop;
- *Silent Data Corruption (SDC)*, where the execution finished but produced an incorrect result – e.g., an error affects the produced data without causing a crash or an abnormally long execution.

The results showed that, in most cases, the error did not affect the outcome of the execution at all (*success*). From the cases where the error actually has an impact on the execution and its outcome the most common outcome is a *crash* followed by *SDC*. Only in some cases an abnormally long execution without crashing was observed (*infinite*).

Schuster et al. have also evaluated the impact of soft errors albeit focusing on errors affecting the control-flow [139]. The experiments showed that, for errors affecting the control-flow of quicksort [56], in most cases (90.96%) the errors resulted in a *crash*. In 5.95% of the cases, an *SDC* occurs. In 2.51% of the cases, the error had no impact on the execution at all (*success*). Similarly to [37], only in very few cases was an abnormally long execution (*infinite*).

In summary, using the terminology defined by ISO 26262 [72] – the safety standard for the automotive industry, the impact of soft errors in software and system levels result in the following possibilities:

- *Intended operating mode*: the intended functionality is provided by the system either because:
  - *Fault masked / no effect*: the fault was masked before becoming an error;
  - *Error handled*: the fault became an error which was handled by a safety mechanism without affecting the intended functionality;
- *Degraded operating mode*: the error is handled by a safety mechanism but the system goes into a fallback mode with degraded performance while guaranteeing the safety critical functionality;
- *Failure*: the error causes the system failure by deviating from the

intended functionality either by:

- *Incorrect functionality*: the system eventually responds but behavior differs from the intended functionality;
- *Irresponsive*: the system does not respond.

2.5. Summary

This chapter investigated the impact of soft errors, the most common type of random hardware faults, on the NoC interconnect and on the software execution in real-time mixed-critical systems. The evaluation considered the stringent requirements of safety standards, such as ISO 26262, which require all possible faults and errors to be taken into account. The investigation included an FMEA, a bottom-up approach that systematically uncovers all impacts of errors and that is usually required as proof by certification processes of safety standards, as well as relevant sources in the literature.

Soft error impacts on the system can range from masked, to wrong output, up to a complete system crash. Timing wise, errors can delay the output of an application, independently of output corruption, to the point where it becomes irresponsive. Based on the insight gained with the investigations a strategy can be employed to efficiently attain dependability against the uncovered threats. The NoC, the central interconnect of many-cores and modern MPSoCs, plays an important role. Unhandled random hardware faults can cause not only data corruption and packet loss and derouting but can present static effects that lead to continuous corruption and blocking scenarios during runtime. Moreover, errors can propagate and indirectly affect the background traffic, propagating through criticalities and throughout the system. Such error effects not only cause the *violation* of integrity and resilience requirements and thus the failure of the system, but also prevent the use of techniques with redundancy in time, such as retransmission and re-execution. In the next chapters, a set of techniques are presented and evaluated that tackle random hardware faults in cross-layer approaches.

3. Designing a Resilient NoC

Regulated by safety standards [72, 35, 69], mixed-critical real-time embedded systems must meet strict *real-time*, *resilience* and *integrity* requirements. In such a context, threats to the intended functionality of the system must be detected and handled appropriately to meet the specified requirements. In case of errors, threats *must* be detected and contained to ensure integrity; a recovery *might* be performed if possible and if resilience is required to reach high reliability levels; and *must* do so in a timely and predictable manner under real-time constraints. Due to the central role of NoCs and their importance to the performance and dependability of the system, their architecture and design is essential to enable their operation under guarantees in the real-time mixed-critical domain.

This chapter proposes a NoC that satisfies strict integrity, resilience and real-time requirements. In contrast to the state-of-the-art, all possible impacts and durations of error effects are taken into account, as uncovered in Chapter 2. The challenges are separated and handled in different network layers [147], as illustrated in Figure 3.1. Errors affecting the NoC's control logic and data are handled in the lower layers. That results in a highly available but lossy service to the upper layers, as error effects are restricted to packet corruption, loss or short delay. Guaranteed integrity and packet delivery of the transmitted data are selectively provided in the upper layers by means of ARQ-based protocols.

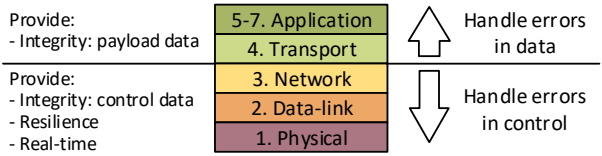


Figure 3.1.: OSI stack overview: errors affecting the control of the network are handled in the lower layers; errors affecting data/payload are addressed in the upper layers.

ARQ-based protocols operating in the transport layer have been recently formally analyzed for real-time NoCs. As shown in Chapter 4, ARQ can be used without jeopardizing the predictability of the system. However, it does require the underlying network to limit the effects of errors and rule out static effects. To achieve that, the proposed approach exploits well-known fault containment and retransmission techniques together with the proposed resilient router design and resilient VC flow-control. The VC flow-control manages the access to VCs in wormhole-switched NoCs. It is a major contributor to cases of static effects due to a dependency on the state of neighboring routers that only becomes evident in case of errors [129]. To overcome those issues, a resilient VC flow-control is also proposed. As a result, the proposed NoC operates under soft errors with formal guarantees.

The remaining of the chapter is organized as follows. After a review of relevant related work in Section 3.1, an overview of the proposed transparent protection is given in Section 3.2. Then each mechanism is addressed individually in Sections 3.3, 3.4 and 3.6 with a brief discussion inbetween in Section 3.5. The evaluation is presented in Section 3.7. Finally, Section 3.8 summarizes the chapter.

3.1. Related work

Fault tolerance approaches for NoCs have been focusing on the requirements of general purpose and high performance computing systems [160, 16, 81, 116, 82, 49, 80]. They usually consider that soft errors cause packet corruption, loss or derouting, and that those effects are transient. However, according to results of the FMEA introduced in Chapter 2, besides the standard effects usually considered in the literature, soft errors can have static effects leading to continuous corruption and blocking scenarios during runtime. Moreover, errors can propagate and indirectly affect the background traffic [128, 129, 124]. Static effects cause the *violation* of integrity, resilience and (real-time) predictability requirements and, thereby, the failure of the system.

In safety critical real-time systems, non-transient errors have a fatal impact on the latency and, by extension, the predictability, leading to the failure of the entire system. Figure 3.2 illustrates the latency of a traffic stream observed over time on two different NoCs: a resilient predictable

NoC; and a baseline NoC which is non-resilient and only predictable in the absence of errors. The latency does not include retransmission. Both NoCs operate correctly and present latencies within maximum l^{max} and minimum l^{min} bounds until soft errors occur at time $t1$ and $t2$. In the resilient design, the effect is transient and its impact is bounded (l^{err}). In the baseline, soft errors cause static effects that result in very high latencies (L). After the error at $t1$, which could be caused by the derouting of a packet from another traffic stream, the non-resilient design is able to recover. After $t2$, however, it remains blocked: affected packets accumulate in the routers' buffers and backpressure propagates throughout the network, leading to permanent blocking of the NoC. The NoC design must rule out static effects without triggering a network reset or handling them as hard errors and ensure controlled impacts (l^{err}).

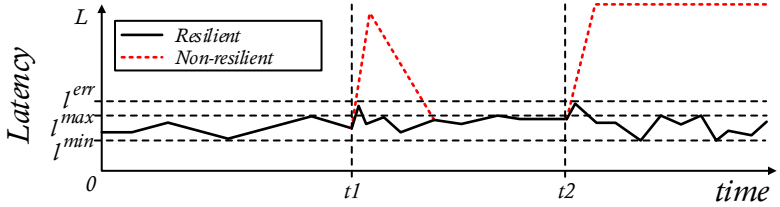


Figure 3.2.: Performance of a traffic stream in a resilient and a non-resilient NoC over time.

The objective of existing approaches is to increase the overall reliability of the network. Most of the work [82, 49, 80] target packet-switched networks, providing reliability and guaranteed delivery of packets in the lower network layers based on hop-by-hop retransmission (between adjacent routers) [147]. Recovering from packet corruption and loss consists of retransmitting a correct copy from the previous router. The task of handling packet derouting is usually delegated to the dynamic routing, which delivers the packet through a different route after the packet has derouted. Variants with end-to-end retransmission and/or forward error correction and hybrids between both are also possible. In forward error correction [147], Error-Correcting Codes (ECCs) are employed to correct the data at the destination instead of retransmitting. Faster wormhole-switched networks have also been similarly addressed [81, 116].

The current state of the art in fault-tolerance for NoCs is well suited

to increase the overall reliability of the network but does not satisfy the requirements of high assurance real-time systems. This is because the techniques either:

- Rely on dynamic routing [82, 49, 80]: the local routing decision violates the real-time requirements since it severely impairs the predictability of the NoC. This is further aggravated by traffic deflection, which allows unexpected traffic overhead in the network – predictability requires static routing [103, 125];
- Address packet-switched or wormhole-switched NoCs with dynamic VC allocation [81, 116]: traffic from different classes in real-time systems requires sufficient independence, i.e., isolation between different traffic classes, since they have potentially different treatment, e.g., priorities – this requires static VC allocation [103, 129];
- Use an insufficient error model [16, 160]: it has been shown that transient faults can lead to static effects [129], leading to unexpected blocking scenarios during runtime – all possible impacts and durations of error effects must be taken into account.

To overcome the gap between resilient NoCs and real-time systems, the NoC should target high dependability systems from the very beginning, as is the case of the IDA NoC [103, 7]. As a starting point, the NoC should be predictable and satisfy the real-time requirements in the absence of errors. Then, the appropriate mechanisms must be introduced to attain the desired dependability.

3.2. Overview of the proposed architecture

To attain high dependability and meet resilience, integrity and real-time requirements, the resilient NoC approach starts by subjecting the NoC to an FMEA-based analysis of [129]. The results of that analysis was introduced in Chapter 2, including a comprehensive error model. The baseline NoC is then hardened against the identified vulnerabilities to soft errors. Those vulnerabilities are addressed by three mechanisms: Fault Containment (FC), Resilient Router Design (RR), and Reliable Transport (RT). These three mechanisms operate in different layers of the NoC and allow to address the requirements of high assurance systems as shown in Table 3.1.

Table 3.1.: Requirements addressed in the different layers of the NoC

OSI Layer	**Requirements**		
	Integrity	*Resilience*	*Real-time*
4-7. Transport - App.	RT		
3. Network	FC		FC
2. Data-link	FC	RR	RR, FC
1. Physical		RR	RR

Fault containment is responsible for ensuring the integrity of the packets in the network. It is also responsible for containing the error to the affected router, preventing its propagation throughout the network and ensuring the predictability of the NoC. The policy for fault containment is packet dropping. Whenever a corrupt or derouted packet is detected, the packet is dropped. A distinction between the integrity of the packet's routing data and the integrity of the payload exists. The routing data's integrity is checked on a hop-to-hop basis, the payload's integrity is checked on an end-to-end basis, since the payload is only relevant upon its delivery.

The resilient router design is responsible for limiting the effects of soft errors in time, ensuring that resilience and predictability are satisfied. Whenever an error affects a component in the router, its resilient design ensures that the component will recover in a bounded period of time.

The reliable transport of data is then responsible for guaranteeing the packet delivery and payload integrity, since packets may be dropped due to errors. The reliable transport is flexible and can be implemented to operate transparently in the transport layer or explicitly in the layers above. An example of the latter is a hardware component such as the Direct Memory Access (DMA) controller, which can implement its own protocol.

The mechanisms introduce new components to the NoC routers extending the IDA NoC architecture. The extension is depicted by the highlighted elements in Figure 3.3. In the sequel, each mechanism is addressed individually.

3.3. Fault containment

To contain the propagation of an error-affected flit, the router is equipped with ingress filters in its input ports, as shown in Figure 3.3. The filter is

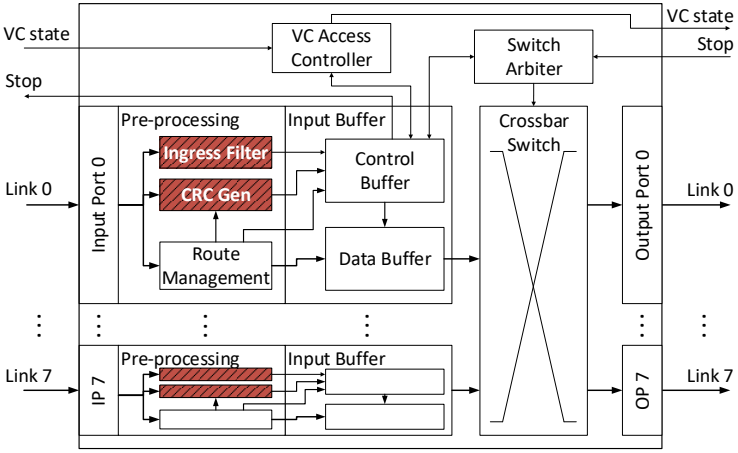


Figure 3.3.: The resilient router architecture.

HF/SF	VC	Flit Type	Route	Tile Port	CRC	Payload
Size	3 bits	2	30	3	3	99
BF/TF	VC	Flit Type	CRC	LOP	Payload	
Size	3 bits	2	2	3	130	

Figure 3.4.: Single Flit (SF), Head Flit (HF), Body Flit (BF) and Tail Flit (TF) formats.

responsible for deciding whether the flit is valid and may be safely propagated or not. If not, the error-affected flit is dropped before altering the router's state. It executes in parallel with the existing route management logic.

The ingress filter must contain flits affected by errors in the upstream router. This boils down to detecting corruption and derouting, similarly to [80]. At the routers, only the routing data (flit header) is checked against corruption.

3.3.1. Containing corrupt flits

To detect corruption, the flits are equipped with Error-Detecting Code (EDC). EDCs, such as parity bit and Hamming code, differ on their detection capability. Cyclic Redundancy Check (CRC) is chosen due to its error detection capabilities that cover not only uncorrelated bit flips but also bursts of errors, a typical effect of crosstalk (corruption in links). The 3-bit CRC generator polynomial 0x5 = $(x^3 + x + 1)$ and the 2-bit 0x2 = $(x^2 + 1)$ are employed. They are able to detect a single bit flip or a burst of up to 3 and 2 erroneous bits, respectively [83]. That suffices given that the CRC is expected to be checked before a second error occurs. Applying other EDCs or other CRC polynomials at design time is also possible. Configuration registers in hardware for the CRC are not required.

Figure 3.4 shows the CRC bits added to the flits, in the first phit of HF/SFs and in the second phit of HF/SFs. Only the flit header is protected because it allows corruption to be detected without requiring the whole flit to be received and processed. Detection can be performed in parallel with route management and can effectively prevent the flit from taking part in the next arbitration cycle without additional delay or stage in the pipeline.

Due to the route management at each router, the CRC code of HF/SFs must be updated accordingly. The new hash is calculated by CRC Gen (see Figure 3.3) after the route has been updated and must be ready only when the flit is leaving the buffer. When the flit transmission starts, the old CRC hash is overwritten with the new one. BF/TF headers are not modified, thus their CRCs are not updated.

In addition to the cases concerning corruption due to transport, corruption during processing must also be accounted for. Regarding the transmission of an already-forwarded flit, the CRC stored in the buffer must be invalidated after being used, so that, in case the same flit is erroneously forwarded a second time, it may be detected and contained. Regarding the route management, the CRC must be calculated based on a trusted copy of the updated route.

3.3.2. Containing derouted flits

The detection of derouted flits is not trivial because one must distinguish between a correctly routed flit and a flit forwarded in the wrong direction, despite their correct header routing data. To distinguish them, the detec-

tion checks the Last Output Port (LOP) traversed by the flit. Although the principle is the same, due to their different formats HF/SFs and HF/SFs are addressed separately below.

In case of a HF/SF, the flit contains the route encoded as a series of runs (output port and a hop counter). At the input port, the ingress filter will check whether the current input port is the one connected with the output port requested in the last router, the LOP. For instance: the SF's requests the output port "North" (N) and in the next router is processed at the input port "East" (E), which characterizes a derouting, since it should be processed at an input port "South" (S). The input port knows at design time to which output port it is connected – e.g., S connected to N, E to W, and so on. The LOP can be found in the first or in the last position in the route, due to route rotation, and is determined at run-time.

In case of a BF/TF, the checking works in the same way but a field containing the LOP is added to the flit header (see Figure 3.4). The field is updated at each hop (similarly to the above-mentioned CRC update): the LOP stored at the input buffer overwrites the field in the flit when the flit is being transmitted. The flit's LOP field does not need to be covered by the CRC because, in the event of an error corrupting the LOP field, it will not be valid at the next input port and the flit will be dropped. This way, the CRC of the BF/TF remains unchanged along the route. Notice that the correct operation of the mechanism requires the integrity of the LOP stored at the input buffer, which can be achieved, e.g., by storing the value twice, one copy for the flit header and another actually for routing purposes.

Uplinks are a special case. When the flit goes from the network interface into the network, no checking is necessary since no derouting is possible so far. When the flit leaves the network and reaches the network interface, the network interface checks whether the last requested port is equal to its port number.

3.4. Resilient router design

To achieve a resilient router it is necessary to eliminate scenarios where soft errors lead to static effects. Although this thesis prefers discussing the architecture, the design plays an important role to achieve resilience. The approach requires the components to be designed in a way that they

continue responding in each arbitration cycle independently from the success of the previous arbitration cycle. Next, each component is addressed separately.

3.4.1. Pre-processing

The pre-processing contains the route management and the newly added CRC generator and input filter, discussed in the previous section. A soft error in the pre-processing may cause flit corruption and derouting. The effects are transient since the state is reset at the arrival of each flit.

3.4.2. Input buffer

The component is responsible for receiving and storing flits, and in parallel, interacting with the arbitration logic and forwarding flits. A soft error in the input buffer may cause transient flit corruption, loss, and derouting.

Soft errors cause static effects in this component when implementing the buffer as one memory (Data Buffer in Figure 3.3). This is the case, for instance, when optimizing the design for FPGAs. In such a design, the VC queues are stored in the memory and each queue is managed through its read and write pointers. The data required for routing decisions (e.g., output port, flit type) are additionally kept in a queue in the Control Buffer due to the limited access to the memory. The static effect (corruption) happens when the control and data buffers have two pointers and an error causes them to desynchronize or to continuously access the memory with a wrong offset. To prevent this, the Control Buffer must store the pointer and the Data Buffer derives its pointer from that one when required. Similar attention is required when handling the flit as phits. In addition, flits received when a VC queue is full must be dropped while keeping the queue data and state unaltered.

3.4.3. Crossbar switch

The crossbar switch connects input buffers to output ports. The connections are configured by the switch arbiter. A faulty crossbar causes flit corruption, loss, and derouting. The effects are however transient, since the crossbar state is reset at each arbitration by the switch arbiter.

3.4.4. Switch arbiter

The arbiter has two stages [99]. The first stage arbitrates in parallel, for each output port, one input buffer that requests access. The second stage, in each input buffer, arbitrates a flit from one of the VCs that received a grant from the first stage. Each stage arbitrates following the Round Robin policy. In wormhole switching, the arbiter is also aware of the VC reservation states managed by the VCAC.

An error affecting the arbiter may cause flit derouting, flit loss, and priority loss. Derouting and loss occur when an error corrupts the arbiter when a grant is being accepted or when it is configuring the crossbar. Those effects are guaranteed to be transient because the grant and configuration data is new at each arbitration cycle. Priority loss occurs when an error affects the state of a round robin arbiter.

A faulty round-robin arbiter may cause priority loss. To ensure that the priority loss is transient (i.e., transient blocking instead of static), the arbiter must detect invalid priorities and advance to a valid priority. Encoding priorities (Most Recently Served) as one-hot already serves the purpose: when no bits or more than one bit is "hot", the priorities are reset to an initial state, ensuring a resilient arbiter.

The same applies to priority-based arbiters, where VCs have different priorities. In that case, an error affecting the arbiter may similarly cause flit derouting, flit loss, and priority loss. Those effects are transient since the grants and requests per priority are new at each arbitration cycle. However, depending on the tie-breaking policy implemented by the arbiter (e.g., round-robin arbitrates requests of same priority to the same output port), the considerations above must also be taken into account.

3.4.5. Virtual channel access controller

The VCAC manages the access to VCs at each output port[1]. Due to wormhole switching, an input buffer has exclusive access to a VC at an output port, creating a "hole" that starts with the first flit of the packet and closes with the last one (the "worm"). A fault affecting the VCAC leads to an improper VC release or an improper reservation. Both may

[1] Alternatively, one might consider the VCAC to manage the access to VCs on links. Both perspectives are equivalent.

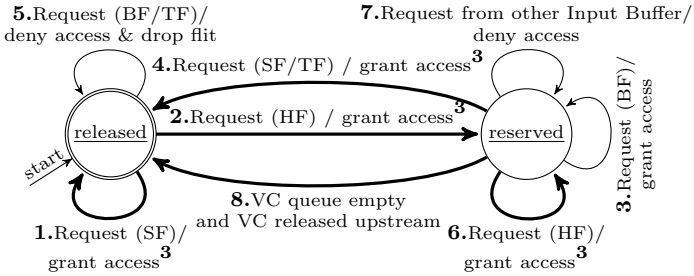


Figure 3.5.: The Resilient VC Flow Control: Mealy state machine.

result in static blocking in the router. An improper VC reservation leads to packet blocking ranging from transient to static since it can only be detected upon arrival of a new packet from a certain direction, which is not guaranteed, or by knowing the state of the upstream routers.

To prevent blocking effects, a *resilient virtual channel flow control* is proposed. Figure 3.5 presents the state machine defining the resilient virtual channel flow control. The state machine is an extension of the state machine presented in Section 2.3.4 (Figure 2.4). The extension is depicted with bold lines. The transitions are shown in the Mealy format: *condition/output*. Detecting an improper VC reservation requires either a timeout or knowing the state of the upstream router. Since timeouts for managing virtual channels are not really an option[2], additional wires between routers are adopted. The scheme requires one wire per VC that implements wormhole switching to inform a router of the VC reservation state of the upstream router.

The regular VC reserve/release operation in Figure 3.5 comprises the transitions 1 to 4, as well as transition 7, which refuses access to a VC reserved to another input buffer. An example is given in Figure 3.6b: first, packets in input N and W request access to the same VC in output E; N wins the reservation first and its flits are being transmitted (trans. 2 and 3) while the packet in W waits (trans. 7); finally, the transfer is completed, the VC is released (trans. 4), and W may continue. In

[2]For instance, finding a suitable timeout value, which depends on the traffic. Values too large result in high latencies; values too small result in packet loss/corruption.

[3]Transitions 1, 2, 4 and 6 also release in all output ports reservations of the respective VC to the input buffer in question.

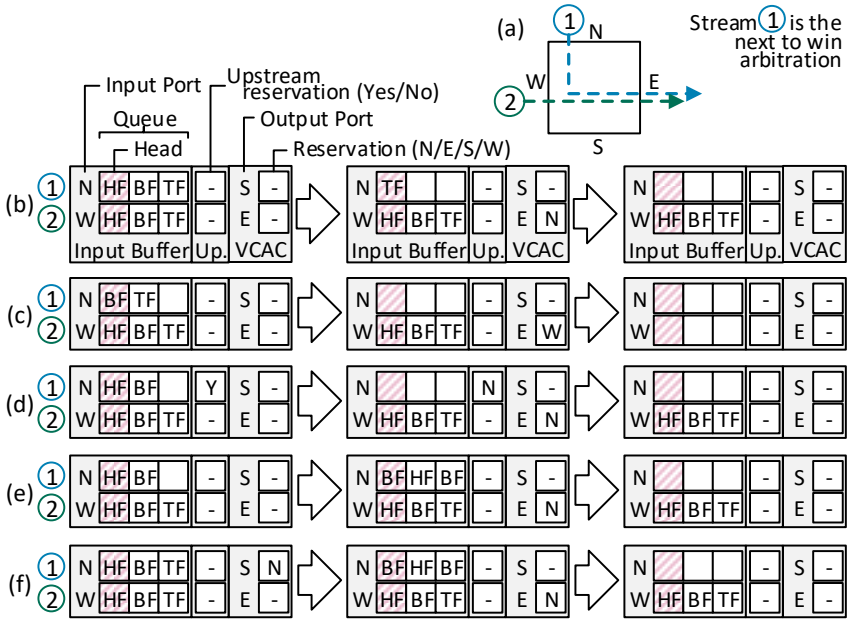


Figure 3.6.: Examples of resilient VC reservation handling, considering one VC and two streams traversing the router as shown in *a*. *b*: regular operation; *c*: improper release; *d*,*e*,*f*: improper reservation.

addition to the regular behavior, the state machine specifies the behavior for faulty scenarios, which are divided in two categories: improper release and improper reservation.

"Improper release" represents cases where flits without routing data are at the head of the VC queue at the input buffer and no VC is reserved for it. These are handled by transition 5, which signals the input buffer to drop the flit. Examples are head-less packets and premature VC releases caused by transient faults. The former is shown in Figure 3.6c, where the flits of the head-less packet in N are discarded.

"Improper reservation" represents cases where the VC is not correctly released. Upon an improper reservation, one of two scenarios can occur: either the VC queue in the input buffer for the improperly reserved VC

is empty or it is not empty. Transitions 4 and 6 handle the case where the queue is not empty: a HF/SF tries to reserve a VC and it is already reserved for his own input buffer, the reservation is handed over to the new packet and the router resumes regular operation (trans. 3 or 4). Transition 8 handles the case where the queue is empty: the VC is released if it is empty and the VC upstream is released. If the VC is reserved upstream, no safe assumption can be made since the rest of the packet may be still coming.

Figures 3.6d and 3.6e show examples for transitions 8 and 6, respectively. In 3.6d, a tail-less packet in N is forwarded and the queue is empty afterwards; when the upstream reservation is released, the respective VC is also released. In 3.6e, a tail-less packet in N is followed by a second packet, which takes over the reservation (trans. 6) and resumes regular operation. In addition, transitions 1, 2, 4 and 6 must also release the VC in all output ports where they are reserved for the same input buffer. Figure 3.6f shows an example for such a case, where a VC is improperly reserved in output port S. The reservation is released by the packet in N (trans. 2) before resuming regular operation.

With the proposed resilient virtual channel flow control, soft errors result only in transient effects since they will last only until one of the transitions 1, 2, 4, 5, 6 or 8 occur.

3.4.6. Link

The link has control and data signals that can be affected by faults. Soft errors affecting the data signals cause flit corruption, which are detected in the ingress filter of the next router. Soft errors in the control signals may cause one-flit loss or blocking lasting one arbitration cycle, depending on whether the valid signal or the flow-control is affected.

To prevent multi-flit loss due to synchronization issues when transmitting flits as phits, a *start-of-flit* signal must be employed instead of a simple valid signal. In the former, the signal is enabled only when transmitting the first phit of a flit, while in the latter the signal is enabled during the transmission of all phits.

The concept is illustrated in Figure 3.7, where a flit f is composed of 4 phits ($f.1$ to $f.4$) transmitted subsequently on the Data bus. The first phit of a flit is highlighted with thick lines. In regular operation, Figure 3.7a,

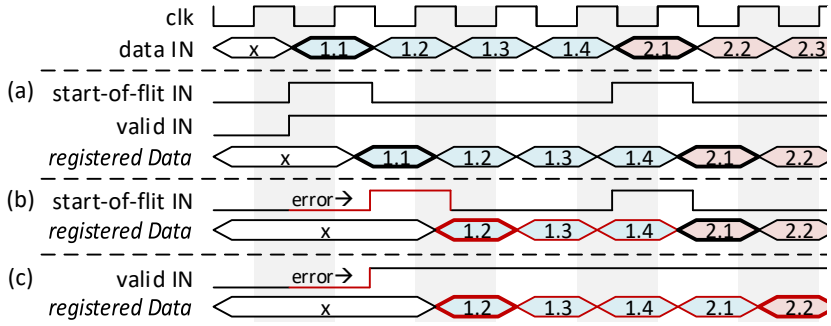


Figure 3.7.: Comparison of link input signals and the registered data using start-of-flit and valid signals. *a*: regular operation; *b*,*c*: error cases.

start-of-flit and valid signals have the same effect: the flit is received as intended by the sender, the *registered data*. In case of an error causing the start-of-flit to be set (or propagated) high one cycle later, only the affected flit will be lost when transmitting flits back-to-back. This is seen in 3.7b where the first received flit starts with the second phit but the subsequent flit is received correctly. However, in the case of the valid signal, it is not possible to identify the end of a flit and start of the next one. This is seen in 3.7c, where first received flit starts with the second phit and includes the first phit of the second flit, since the receiver expects four phits after the valid signal is first set high. The same occurs to the subsequent flits, resulting in the loss of all flits transmitted back-to-back on that link after the error. Corrupt flits are discarded by the ingress filter (cf. Section 3.3 and Section 3.6).

Faults affecting the VC reservation signals introduced above may cause the loss of a packet when it enables the transition 8 in Figure 3.5.

3.5. Between lower and upper layers

Before introducing the reliable transport, this section discusses what has been achieved with the hardening so far.

As summarized in Table 3.1, the proposed fault containment ensures: *integrity* of the data utilized in the lower layers of the NoC (the flit header);

and part of the *real-time* requirement as it prevents the error from propagating to other traffic streams and to other routers. The resilient router design ensures the *resilience* and the remaining part of the *real-time* requirement by ruling out static effects and providing for a predictable recovery that can be limited in time.

The behavior of the network *before* applying the reliable transport can be summarized as follows. On the end-to-end perspective at the NI, a packet sent through the NoC and affected by an error is either:

1. delivered correctly,
2. delivered correctly with a small delay l^{rec},
3. delivered with corrupt payload,
4. or dropped/lost.

The reliable transport is responsible for guaranteeing the packet delivery and guaranteeing the integrity of the transported data – i.e., the payload.

3.6. Reliable transport

Aiming at flexibility, the approach provides guaranteed delivery and payload integrity on an end-to-end basis. Thus, the reliable transport is implemented in the transport layer or above. It relies on EDCs or ECCs for error detection and possibly correction and relies on protocols such as ARQ and multipath routing for guaranteeing packet delivery.

The approach enables the system controller or application to configure at runtime in the NI the appropriate configuration (combination of protocol+EDC/ECC) for each transaction according to its characteristics, reducing unnecessary overheads, both in terms of traffic and power consumption. For instance, a DMA transfer may use its own optimized protocol, while a command to an actuator could use Stop-and-Wait ARQ. When a retransmission cannot be afforded due to a deadline miss, multipath routing can be employed. Besides, periodic sensor readings may require only integrity guarantee with EDC/ECC and no delivery guarantee. Moreover, the above-described, diverse traffic may co-exist in the NoC.

Formal guarantees for ARQ-based protocols operating in the transport layer of wormhole-switched NoCs will be introduced in Chapter 4. Three protocols will be addressed: Stop-and-Wait, Go-Back-N and DMA ARQ,

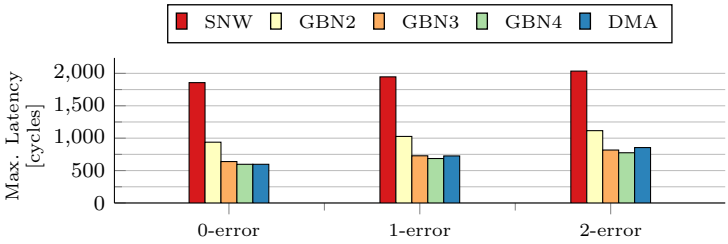


Figure 3.8.: Maximum latency of an 8kB DMA on different ARQ protocols and error scenarios (cf. Section 4.4).

a protocol optimized for DMA transfers. Formal guarantees for other protocols, such as multipath routing, can be similarly derived. As will be seen in Chapter 4, the latency overhead introduced by the protocols' handshaking is acceptable and they perform very well with typical real-time on-chip traffic, provided that the appropriate configuration is employed. As an example, the latency of an 8kB DMA transfer when using different protocols and configurations is plotted in Figure 3.8 (further details on the setup in Section 4.4). In the figure, in the error free case (0-error), the handshaking overhead of ARQ is almost negligible for Go-Back-N with $N = 4$ (GBN4) and the DMA ARQ (DMA).

To provide minimum performance guarantees, formal analyses assume a given number of errors that the system may experience in a given time interval, the so called k-error scenario [130, 13]. In the NoC, it translates to errors affecting a given packet or transmission. k depends on the BER to which the NoC is subjected and can be calculated using methods, e.g., in [67]. k must be selected so that the probability that the packet or transmission experiences more than k errors, i.e., a failure, is negligible according to compliance levels of safety standards [72, 69, 35]. The k-error scenario is then used to calculate the worst-case latencies and response times of the system under errors. Usually, at most one error will be considered since the probability of two errors within a hundred clock cycles is negligible. Finally, the end-to-end latencies in the NoC under errors depend on the specific protocol.

Figure 3.8 shows latencies in scenarios with 0, 1 and 2 errors. Error occurrences imply slightly longer latencies due to timeout triggering the retransmission and the retransmission itself. Nonetheless, that does not

prevent the use of ARQ in a real-time system. Alternatively, the error case can be modeled as an overload scenario in typical worst-case response time analysis [57]. However, exploring that idea is out of the scope of this work.

N.B. regarding the delay l^{rec}: when employing retransmission protocols, it is not necessary to know the exact value of l^{rec}. The formal guarantee under errors only requires l^{rec} to be smaller than the time to retransmit the packet (see Chapter 4).

3.7. Experimental evaluation

The resilient NoC was evaluated with respect to reliability, performance under errors and implementation overhead when compared to the baseline NoC. The performance under errors is evaluated with synthetic random traffic as well as a real-world use case. The evaluation considers a NoC in a 2D-mesh topology with different sizes, 5 VCs, and one up-link per router. From the reliable transport layer, only EDCs are employed to ensure the payload integrity.

The objective of the experiments is to evaluate the predictability and reliability of the NoC under soft errors. Thus, impact of bit-flips in the NoC or the performance of ARQ-based protocols are not addressed here. On the impact of soft errors in NoCs, the interested reader can refer to Chapter 2. Transport protocols have well know properties and can be applied on top of the resulting NoC. On that topic, the interested reader can refer to Chapter 4.

3.7.1. Reliability

First, the reliability metric $\mathcal{R}(t)$ is evaluated, which is the probability that the NoC does not fail during a time interval $[0, t]$ [67]. In a high dependability mixed-critical real-time system, the failure is defined as the violation of integrity, resilience or real-time latency guarantees due to errors, including static effects leading to blocking. Packet loss is not considered as a failure since it is handled in the transport layer. In practice, expected BERs[4] are in the order of 10^{-12} bit-flips per hour [6]. The design must however

[4]BERs derived for sequential and combinational logic [61] with data from [6] for 65nm CMOS SRAM. Masking effects [41] are not taken into account.

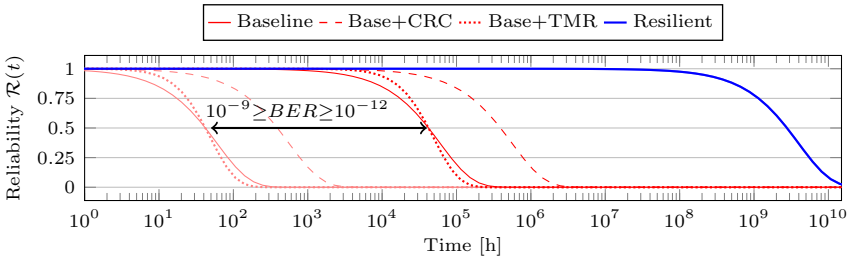


Figure 3.9.: Reliability comparison of the resilient NoC and non-resilient ones. A 5x5 NoC size is considered.

consider higher rates as a safety margin [91]. Here, BERs from 10^{-12} up to 10^{-9}/hour are considered. Additionally, a permanent fault rate of 10^{-11}/h per router[5] is considered.

Figure 3.9 plots the analytical $\mathcal{R}(t)$ for the non-resilient, baseline NoC and the proposed, resilient NoC considering a 5x5 2D-mesh topology. Two variants of the baseline NoC with CRC checking (Base+CRC) and with TMR (Base+TMR) are also plotted. The TMR is non-reparable with an ideal voter [67]. It is clear that, independently of the BER, regular non-resilient NoCs are limited in time due to errors from which they cannot recover. Employing ingress filters (Base+CRC) improves the reliability but it continues limited. Moreover, contrary to common sense, triplicating the NoC does not make it more reliable with respect to soft errors. In time, the more-than-double area overhead of the TMR has more impact than the initial advantage of withstanding one error. The proposed hardening enables the NoC to drastically increase the reliability in time by appropriately containing soft errors and recovering from them.

Notice that the reliability of the resilient NoC in time, plotted in Figure 3.9, does not vary with the BERs. That is a consequence of the high resilience to transient faults. Since the probability that the NoC fails due to transient faults is so small, permanent faults become the dominating factor independently of the BER. In fact, at about 10^8 hours, the occurrence probability of a permanent fault becomes significant and the reliability starts decreasing. Notice that for the other NoCs (Baseline, Base+CRC,

[5]The occurrence of a permanent fault leads directly to failure. The fault rate per router is inspired from processor failure rates in [111].

Table 3.2.: Comparison of FIT rates

	BER/h	3x3 NoC	5x5 NoC	8x8 NoC
Baseline	10^{-12}	$1.86 \cdot 10^{3}$	$1.65 \cdot 10^{4}$	$1.16 \cdot 10^{5}$
	10^{-9}	$1.86 \cdot 10^{6}$	$1.65 \cdot 10^{7}$	$1.16 \cdot 10^{8}$
Base+CRC	10^{-12}	$2.02 \cdot 10^{2}$	$1.78 \cdot 10^{3}$	$1.24 \cdot 10^{4}$
	10^{-9}	$2.01 \cdot 10^{5}$	$1.78 \cdot 10^{6}$	$1.24 \cdot 10^{7}$
Base+TMR	10^{-12}	$2.24 \cdot 10^{3}$	$1.98 \cdot 10^{4}$	$1.39 \cdot 10^{5}$
	10^{-9}	$2.24 \cdot 10^{6}$	$1.98 \cdot 10^{7}$	$1.39 \cdot 10^{8}$
Resilient	10^{-12}	$9.0 \cdot 10^{-2}$	$2.5 \cdot 10^{-1}$	$6.4 \cdot 10^{-1}$
	10^{-9}	$9.0 \cdot 10^{-2}$	$2.5 \cdot 10^{-1}$	$6.4 \cdot 10^{-1}$

and Base+TMR), a failure due to transient faults will certainly occur before a permanent fault occurs.

The FIT metric measures the number of failures in a billion hours of device-hour operation [67]. Table 3.2 reports the FIT rates for different NoC sizes and BERs. Even in small topologies, non-resilient NoCs present very high failure rates. Besides, TMR leads to more failures than no redundancy. To put the values in perspective, high dependability systems (e.g., in the automotive domain) must present less than 10 random hardware failures in time when implementing critical functionality with SIL 4, the highest safety integrity level [69]. In a system providing less critical functionality with SIL 1, the lowest integrity level, up to 10^4 FITs are acceptable [69]. Notice that a final FIT rate for the MPSoC must consider other components, resulting in even larger FITs. As a component of the final system, the MPSoC requires a dependable and resilient NoC.

Despite the NoC's high reliability, errors still have an impact on the traffic latency while the routers recover from them. That impact is evaluated next.

3.7.2. Performance under errors: random traffic

The performance of traffic under errors was evaluated by means of error injection experiments carried out in the OMNeT++ simulator [153]. For that, uniform random traffic consisting of 3-flit packets was injected in the NoC, with the injection rate varying with the experiment. The NoC operates with a clock period of 2ns and has a flit transmission time of 1 clock cycle. The errors listed in Table 3.3, which were derived from the results of

Table 3.3.: Error model

Component	**Fault Effects** on flits			
	Corruption	Loss	Derouting	Blocking
IB	✓	✓	✓	
Crossbar	✓	✓	✓	
SA		✓	✓	✓
VCAC				✓
Link	✓	✓		✓

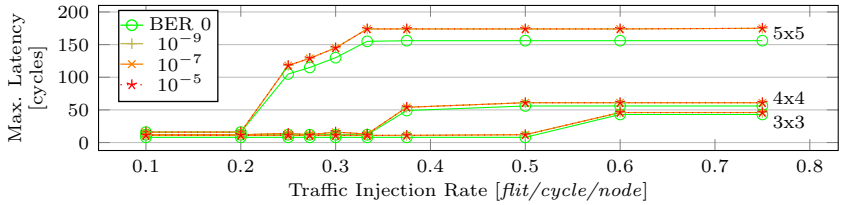
Section 2.3, were injected randomly[6] into the NoC. The occurrence probability of an error depends on the affected data structure and the BER. The employed BERs range from 10^{-9} up to 10^{-5}/h. The high BERs are employed to stress the resilient NoC and evaluate and validate its behavior in extreme scenarios that serve as lower bound for the expected performance in lower error rates. Permanent faults are not modeled.

In the first experiment, faults were injected randomly throughout the entire NoC. Figure 3.10 shows the observed worst-case performance of the proposed NoC, as the load increases, for different NoC sizes, packets sizes and BERs. Performance degradation is minimal even when experiencing high error rates. By design, errors also cause packets to be dropped when corrupted or derouted. On average, $1.11 \cdot 10^{-14}$% of the packets were dropped under BER 10^{-5}, $1.11 \cdot 10^{-16}$% under 10^{-7}, and $1.11 \cdot 10^{-18}$% under 10^{-9}.

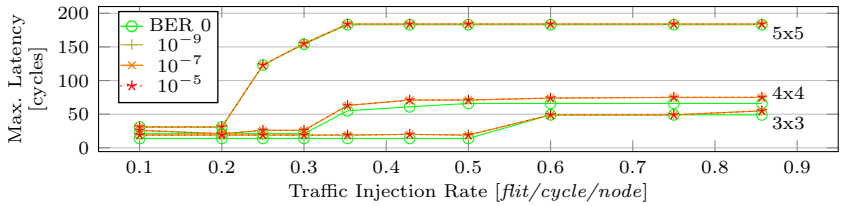
The impact of errors on latency in the hardened NoC depends on the number of errors that affect a packet in the network and on the current load of the router where the error occurs. The load depends on the mapping, the routing algorithm and indirectly on the NoC size – a larger NoC may present higher loads in central routers, up to the maximum load a router is able to handle. This can be seen in Figure 3.10a where maximum latencies both in the error free and in the error cases increase with the load until congestion occurs in the network – e.g., 0.33 in 5x5 NoC. Notice that the latencies reported in Figure 3.10 comprise neither the retransmission nor the time spent in the NI in case of congestion, only the time spent inside the network.

Moreover, the experiment shows that the performance degradation is

[6]To speed-up simulation, errors were injected in active areas without impairing the evaluation.



(a) 3-flit packets



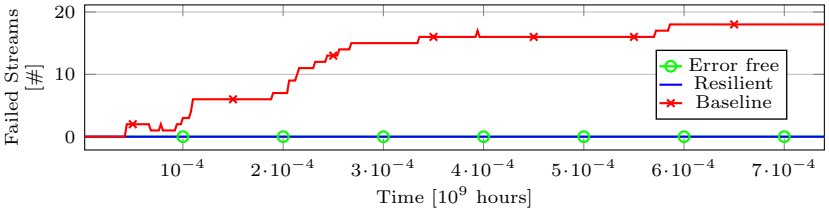
(b) 6-flit packets

Figure 3.10.: Observed worst-case NoC performance under random uniform traffic as the load increases, varying NoC sizes, packet sizes and error rates.

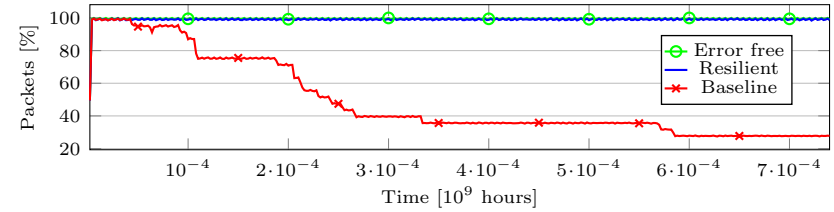
predictable even under very high load and BER and that increasing the packet size does not have major impact on the latency under errors. This can be seen in Figure 3.10b, which shows the NoC performance when doubling the packet size (6-flit packets).

In a second experiment, the error propagation between affected and unaffected VCs were evaluated. Therefore, errors were injected only in the central router and its links. The traffic streams were then classified into *affected*, whose packets traversed the faulty router, and *unaffected* streams, whose packets were not on VCs that were affected in the faulty router. It was observed that error effects do not propagate between different VCs as the latency, integrity and delivery of packets in unaffected VCs are not affected. Considering that different traffic classes and criticalities are allocated to different VCs in a mixed-critical system, that means that *sufficient independence* is achieved in the presence of errors. The same behavior was observed across different network sizes.

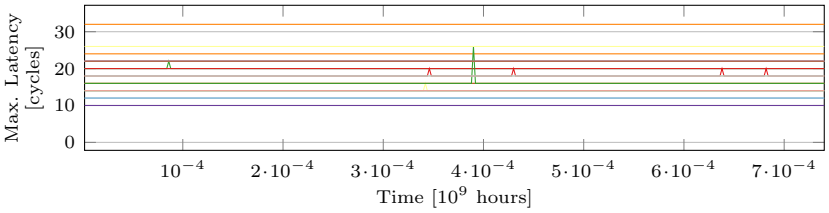
In a third experiment, the performance of the proposed NoC in time



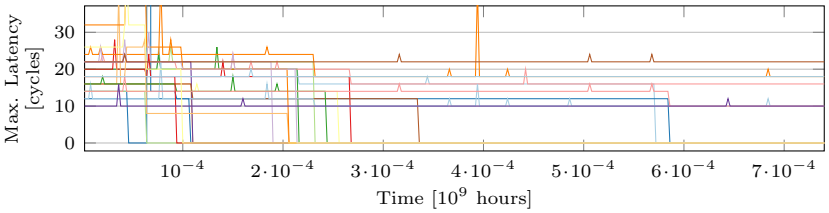
(a) Failure of traffic streams in time



(b) Overall packet delivery in time



(c) Maximum latency in time per traffic stream: resilient NoC



(d) Maximum latency in time per traffic stream: baseline NoC

Figure 3.11.: Performance in time of the resilient NoC and a non-resilient, baseline one. 5x5 NoCs, 3-flit packets, traffic injection rate 0.2 (flit/cycle/node) and BER=10^{-6}.

is evaluated and compared with a baseline one as errors are injected randomly throughout the entire NoC. The results are reported in Figure 3.11, where each plotted point represents a time interval of two thousand hours. Figure 3.11a shows the number of failed streams in time, from a total of 25 traffic streams in the NoC. The definition of failure from Section 3.7.1 is employed. The resilient NoC is able to recover from errors, as it ensures that soft errors have only transient effects. In contrast, the baseline NoC is not able to do so and traffic streams become blocked when certain errors occur. Notice that not all errors lead to a stream failure (see Section 2.3).

The packet delivery is reported in Figure 3.11b. The metric consists of the overall number of packets delivered among all traffic streams in the NoC relative to the error free case (100%). With time, the baseline NoC delivers less packets as a result of blocking and static effects. In the resilient NoC, on the other hand, packets are delivered continuously albeit with expected loss ($1.39 \cdot 10^{-15}$%) due to the forward error recovery nature of the approach.

The maximum latency observed in time in the resilient and baseline NoCs are given in Figures 3.11c and 3.11d, respectively. The latencies are given for each traffic stream, 25 in total (some lines overlap). The predictability of the proposed NoC is seen as the maximum latency varies seldom and within a limited range. In contrast, the unpredictable behavior of a non-resilient NoC presents very high latencies or blocking (latencies equal 0). Moreover, Figures 3.11c and 3.11d show the observed experimental trend that was previously only illustrated in Figure 3.2.

In case of a NoC in a TMR configuration, latencies and packet delivery do not vary under errors. The voter ensures that a packet sent is delivered as soon as its delivery is correctness is confirmed – i.e., the packet is delivered by 2-out-of-3 NoC instances. A longer latency in one of the NoC instances shows that its behavior diverges from the others instances, indicating its failure. That contrasts with the performance of the resilient NoC where errors can cause the latency to vary. However, the unaffected latency and packet delivery hold only as long as the TMR itself does not fail – i.e., 2-out-of-3 instances survive. After the failure, depending on the voter, the same behavior of the baseline NoC is observed.

In addition to the presented experiments, the proposed NoC has been evaluated in many other experiments (thousands), where we vary the packet sizes, traffic patterns, network sizes, and error rates. The experiments em-

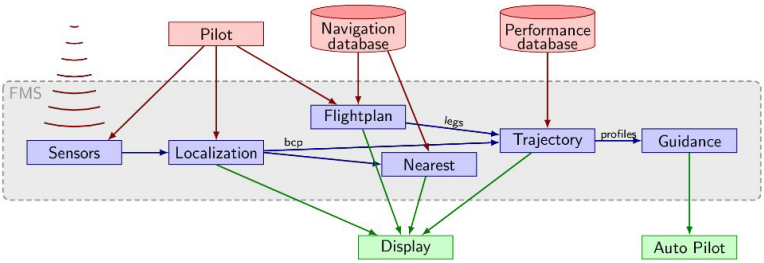


Figure 3.12.: Overview of the FMS application [40].

ployed benchmarks as well as randomly generated mappings and loads, the latter as an effort to stimulate worst-case scenarios. Compared to the results shown here, similar or better performance was observed across all experiments – i.e., no worse case was observed including setups with different traffic patterns and packet sizes.

3.7.3. Performance under errors: FMS use case

The resilient NoC was also evaluated with a real-world use case, Thales' FMS [40]. The Flight Management System is a high dependability embedded application in modern avionics that automates several in-flight tasks. It is widely found in civilian as well as in military aircraft. A functional overview of Thales' FMS [40] is shown in Figure 3.12 (arrows indicate the data flow). The application consists of 6 task groups: *Sensors*, *Localization*, *Flightplan*, *Nearest*, *Trajectory* and *Guidance*. The tasks in the groups are executed periodically, when in automatic mode. Some of them can also be triggered aperiodically through a manual intervention by the pilot. In our evaluation, we focus on the *Localization* task group.

The *Localization* task group is responsible for periodically computing the Best Computed Position (BCP), the most probable position of the aircraft, using data from sensors, such as the Global Positioning System and the Pure Inertia Reference System, received from the Sensors task group. The task group is detailed in Figure 3.13 (arrows indicate the data flow). The BCP computation also must account for settings that can be modified by the pilot, captured in Figure 3.13 by the aperiodic tasks LOC_{A1}, LOC_{A2} and LOC_{A3}. The execution follows the Acquisition-Execution-Restitution

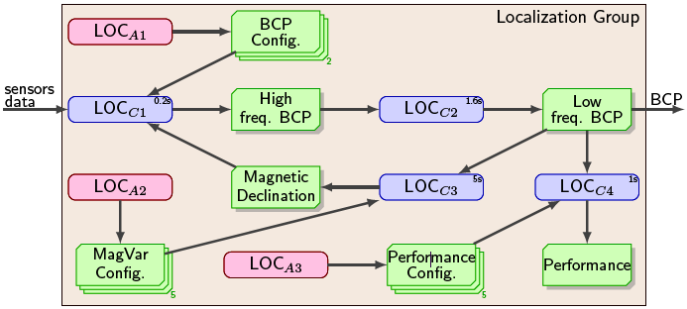


Figure 3.13.: FMS: Localization task group [40].

task model, where at the end of each execution, the output of a task is written to the dependent tasks' buffers [40]. The FMS has, among others, a requirement at the application level stating that a task can execute at most 3 times without receiving new valid input data. In that case, the last valid input data is used. After the third time, the task enters an error mode – i.e., fails. Corrupt data is discarded and not considered as valid.

When mapped to an MPSoC, the tasks' outputs are transferred over traffic streams over the NoC. Table 3.4 list the traffic streams associated with the periodic tasks of the *Localization* task group. For instance, the output of LOC_{C1} is transferred to LOC_{C2}'s buffer through stream $C1 \rightarrow C2$ every 200ms on an 11-flit-long packet. To obtain a safe upper bound on the failure probability, this evaluation considers that each pair of communicating tasks are mapped to the two tiles that lie the most distant from each other, i.e., the corners of the NoC in a 2D-Mesh topology. That results in the longest NoC traversal for each traffic stream.

Since the application tolerates data loss, the traffic streams only employ EDCs to guarantee integrity in the reliable transport – i.e., traffic streams do not employ a reliable transport protocol to guarantee delivery. For other cases, such as application initialization which might require guaranteed data delivery, e.g., through DMA, the interested reader can refer to Chapter 4.

The failure probability of the *Localization* task group due to soft errors in the NoC is reported per stream and per task group in Table 3.4. The failure probability in the resilient NoC equals to the probability that three

Table 3.4.: Performance of localization computing tasks under BER 10^{-6}/h

Traffic Stream	Period [s]	Size (flits)	Failure probability (3 subseq. lost packets)		
			3x3 NoC	5x5 NoC	8x8 NoC
$C1 \rightarrow C2$	0.2	11	$1.31 \cdot 10^{-39}$	$2.57 \cdot 10^{-39}$	$5.64 \cdot 10^{-39}$
$C2 \rightarrow C3$	1.6	11	$1.31 \cdot 10^{-39}$	$2.57 \cdot 10^{-39}$	$5.64 \cdot 10^{-39}$
$C2 \rightarrow C4$	1.6	11	$1.31 \cdot 10^{-39}$	$2.57 \cdot 10^{-39}$	$5.64 \cdot 10^{-39}$
$C3 \rightarrow C1$	5	3	$3.37 \cdot 10^{-42}$	$1.13 \cdot 10^{-41}$	$3.81 \cdot 10^{-41}$
Task group overall:			$3.94 \cdot 10^{-39}$	$7.71 \cdot 10^{-39}$	$1.69 \cdot 10^{-38}$

subsequent packets are lost or corrupt while in the NoC. The evaluation considered a BER of 10^{-6}/h and different NoC sizes, and safely assumed that each stream traverses the longest route in the NoC. Due to the NoC's properties, such as fault containment, and as shown in previous experiments, it suffices to consider errors directly affecting the packets under evaluation, which includes errors affecting the packet contents and registers critical to packet transmissions in the network. The worst-case packet latencies were obtained from the application activation patterns and their mapping in the NoC with [125].

The results per stream and per task group are listed in Table 3.4. The very low failure probabilities per stream and also for the whole task group show that this type of traffic can be safely transported in the network without the use of transport protocols even under high BERs. It results in less traffic in the network, since there is no overhead from transport protocols, and consequently in lower power consumption. This type of traffic is also typically seen in transmissions of periodic sensor readings.

3.7.4. Implementation overhead

The resilient NoC was implemented in VHDL and synthesized in Xilinx ISE targeting a Virtex-6 FPGA (xc6vlx760) and a frequency of 100MHz. The evaluation and comparison involved a 5-port router of the proposed and the baseline NoCs. In comparison with baseline, the router size in the resilient NoC increases 5.39% when accounting for total data stored in the router. Optimized for FPGAs, the data buffers instantiate 5 Block RAMs. Thus, the total data in the router corresponds to bits stored in registers and in the BRAMs. Figure 3.14 details the resource usage and total power for the router and its internal components. In the figure, the

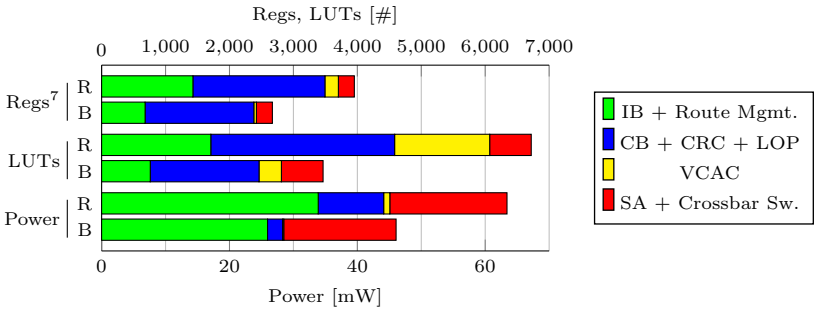


Figure 3.14.: Synthesis results for a 5-port router of the baseline NoC (B) and the resilient NoC (R).

input buffer is divided into two categories: the control buffer and ingress filter (CB+CRC+LOP) and the remaining components of the input buffer (IB). Notice that the 5 BRAMs are not included in the figure.

The implementation overhead of the resilient NoC when compared to the baseline is caused mainly by the ingress filter and by the resilient VC flow control. Indeed, the ingress filter (CB+CRC+LOP) is responsible for most of the additional register bits (43.73%) and 36.06% logic (LUTs). However, such an ingress filter is a standard component in many approaches, such as [80, 82] for NoCs and [98] for (off-Chip) Ethernet, and can be considered as a necessary baseline resiliency cost. The overhead that is unique to the proposed resilient approach is mainly introduced by the resilient VC flow-control. The resilient VC flow-control (VCAC) requires additional wires in the links and control logic, corresponding for 10.40% additional registers and 34.92% additional logic. Other minor overheads are caused, e.g., by the packet-dropping logic in the input buffer (IB).

The energy consumption was evaluated with Xilinx Power Analyzer [156]. Under full load (random traffic with random payload), the resilient router consumes 37.68% more energy than baseline (cf. Figure 3.14). When idle (no traffic), the overhead is 0.28% (not shown in the Figure). Under regular operation, the router load is expected to be closer to the latter than to the former.

Figure 3.15 compares the overhead of the resilient NoC with the TMR

[7]Not including the data buffers, synthesized as Block RAMs.

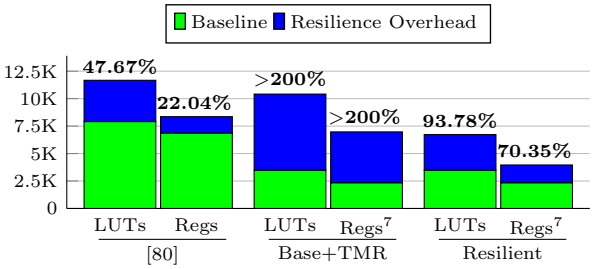


Figure 3.15.: Comparison of resilience overhead per router (Virtex-6 FPGA).

approach and related work [80]. The reader should keep in mind that, *despite* the comparison, the two approaches are not equivalent (cf. Section 3.1). Before discussing the results in Figure 3.15, two considerations are required. First, the baseline router employed in the evaluation is very lean, implemented with the minimal functionality required. Second, the synthesis tool optimally maps the data buffers (28Kb each) to Block RAMs of the FPGA, leading to an apparent large overhead with respect to the register usage. To put it into perspective, our router stores over 17 times more data than [80]'s. Nonetheless, our absolute overhead when synthesized to the same Virtex-6 FPGA family is similar to [80]'s.

Considering the total data stored in the router including data buffers, the relative overhead of resilient is only 5.39% as opposed to [80]'s 22.04%. In comparison to TMR, not only is the resilience approach much more efficient with respect to resource usage (area) and power but also more effective (cf. Section 3.7.1). The resilience approach achieves predictability and much higher reliability with only a fraction (5.39%) of TMR's total relative overhead of >200%. Moreover, the resilient NoC requires at most 37.68% (in a traffic peak) additional power while TMR has a constant twofold power overhead. Besides, TMR implies a substantial increase in interconnecting wires, leading to design routing complications, potential congestion and lower frequencies.

3.8. Summary

This chapter presented a resilient wormhole-switched NoC hardened against soft errors. In contrast to the state-of-the-art, the proposed NoC aims at the integrity, resilience and real-time requirements of high dependability mixed-critical systems and addresses, at the same time, the challenge of silent data corruption. To achieve that, the approach takes into account all possible durations and impacts of soft errors, uncovered by means of an FMEA. Errors are detected and handled by three mechanisms distributed in different layers of the network stack. For the sake of predictability and integrity, error-affected packets are dropped and guaranteed packet delivery is selectively provided on an end-to-end basis. The resilient NoC presents a predictable behavior even under very high error rates with formal guarantees and acceptable hardware overhead, made evident by an experimental evaluation that includes an industrial avionics use case.

4. ARQ Protocols for NoCs

ARQ protocols are widely used in practice to guarantee the delivery of data transmissions over unreliable or noisy communication channels. They can be employed in the data-link layer – over a link between two adjacent network nodes – as well as in the transport layer – between sender and receiver, across potentially multiple networks and nodes [147, 90]. Those strategies are also known as Hop-to-Hop (H2H) and End-to-End (E2E), respectively [13, 130].

The most traditional application of ARQ is on the transmission over a single noisy communication channel – i.e., H2H [90]. In that case, ARQ ensures a high level of reliability across a single transmission hop. As communication lines became less noisy, ARQ protocols have been implemented more often at the edges of the network to provide E2E reliability in the transmission of packets over multiple hops in a network – i.e., over multiple communication channels and network nodes [90]. The basic elements of ARQ protocols consist of: information packets (DATA), which transfer the user packets; control packets – Acknowledgements (ACKs) and Negative Acknowledgements (NACKs); EDCs or ECCs – such as the CRC; and timeout mechanisms. The basic elements are depicted in Figure 4.1.

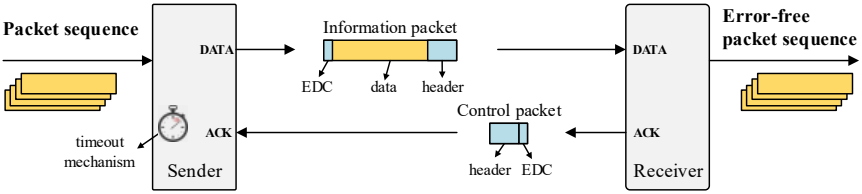


Figure 4.1.: Basic elements of ARQ (adapted from [90]).

This chapter addresses the use of E2E ARQ protocols on real-time NoCs. The remainder of the chapter is organized as follows. Section 4.1 discusses the state of the art, followed by the modeling of E2E protocols on NoCs

with Compositional Performance Analysis (CPA), a formal real-time analysis method, in Section 4.2. Next, Section 4.3 discusses three E2E protocol variations in the context of NoC and formally analyzes them with respect to worst-case latencies with CPA:

- Stop-and-Wait ARQ – the simplest variation;
- Go-Back-N ARQ – an extension of Stop-and-Wait for higher throughput;
- DMA ARQ – variation optimized for bursty DMA transfers.

The first two are classic variations of ARQ, DMA ARQ is introduced in this thesis. Section 4.4 presents the evaluation with simulation as well as analytical results. Finally, Section 4.5 summarizes the chapter.

4.1. Related work

Many mechanisms for increasing the reliability of the NoC have been proposed throughout the years. They have been compared in terms of power and area overheads as well as their resulting reliability improvement. [16, 43] investigate the energy-reliability trade-off between different soft error detection and recovery strategies at the data-link layer. Kim et al. and Murali et al. present analyses of NoC designs with respect to performance, reliability and energy perspective. They allow a good comparison between different design choices and error recovery schemes applied in both data-link and transport layers. The latency estimations however are either based on queueing theory [81] or simulation [104], and thus cannot provide the bounds required in the real-time domain [79].

In real-time systems, formal performance analyses play an important role by guaranteeing responsiveness in time. For NoCs as well as networks in general, formalisms such as Network Calculus, schedulability and response-time analyses, and dataflow analysis can be employed to provide minimum performance guarantees [79]. Those guarantees are usually with respect to latency and backlog, which address performance as well as resource dimensioning.

Network calculus is a mathematical framework for deriving worst-case bounds on latency and backlog in a single node as well as a network of nodes. It is similar to conventional system theory, where a system is described as a set of input, transfer and output functions. In network calculus,

those functions are referred to as arrival, service and output curves, respectively. Its theory was pioneered by Cruz in [29, 30] and further developed by Chang and Le Boudec and Thiran with the *min-plus algebra*, where addition and multiplication from conventional system theory are replaced by minimum and addition, respectively. In the context of NoCs, worst-case flit and packet delays per traffic stream were obtained for wormhole-switched and packet-switched NoCs [119, 120]. Relevant extensions of network calculus are stochastic network calculus [73] and real-time calculus [25].

Schedulability and response time analysis is a mathematical formalism for investigating the timing properties in real-time systems [79]. Originally proposed for the analysis of uniprocessor systems [92], schedulability analysis has been greatly extended over the years to cover more general workload models as well as multiprocessing and NoCs [14, 142]. In schedulability analyses, workloads consist of tasks that are activated according with activation models – e.g., periodic and sporadic are two examples of models, where tasks are released at regular, periodic intervals or released at arbitrary times with a specified minimum distance between activations, respectively. Besides the activation model, tasks have other attributes, such as, the Worst-Case Execution Time (WCET) and the relative deadline – i.e., how much work the task requires and when the work must be finished the latest, respectively. Given the mapping of tasks to resources and their scheduling policies, the schedulability analysis can test whether the system is schedulable or not. Besides testing for schedulability, response time analyses also report metrics such as Worst-Case Response Time (WCRT) and maximum number of pending activations – those correspond to worst-case latency and backlog in networks, respectively. In the context of NoCs, many analyses have been developed, being the most recent ones [125, 23, 71, 70, 158].

Relevant to this work is CPA [62]. The CPA framework has been widely used for modeling off-Chip communication, such as switched Ethernet [31, 148] and CAN buses [11]. CPA is a response-time analysis framework with event models that resemble Network Calculus and it will be introduced with details in Section 4.2.1.

A formal communication time analysis of wormhole-switched NoCs with two-stage arbitration was proposed in [33]. Based on CPA, the analysis provides latency bounds for individual traffic streams on the network. The work has been extended in [125] to support the sharing of virtual channels by traffic streams. The analysis and its guarantees are valid in the real-

time domain but they have the implicit assumption that no errors affect the network, and hence, the transmission.

ARQ protocols have been formally analyzed by Axer et al. for general packet-switched networks. They address the variants Stop-and-Wait ARQ and Go-Back-N ARQ. The CPA-based analysis models the protocols on the transport layer and considers both the error-free and the error cases. The analysis however cannot be directly applied to NoCs, whose more advanced designs feature wormhole-switching. In wormhole-switched networks, packets are composed of flits and the arbitration is performed in a flit granularity [125]. Thus, an integration of the transport layer analysis, operating on a packet basis, with the NoC analysis, operating on a flit basis, is required. To the best of my knowledge, no formal analysis has addressed the use of ARQ-based protocols for NoCs.

4.2. Modeling end-to-end transport protocols

Transport protocols, such as DMA ARQ and Go-Back-N ARQ, introduce an additional flow control to the communication. That comes from packets that are retained, e.g., due to the protocol's handshaking or due to a retransmission – i.e., until ACKs from previous packets are received or until a timeout occurs, respectively. A circular dependency is then formed where the performance of the transport layer depends on the network performance, which in turn depends on the traffic injected by the transport layer. The problem of providing formal performance bounds for such networks is solved here using CPA, which provides for an easy integration of both network and transport layer analyses due to its compositional nature.

4.2.1. Compositional Performance Analysis (CPA)

CPA relies on independent local analyses of system resources, such as router ports and CPUs, and a global analysis loop, which aggregates the local results [62]. It provides worst-case response times and jitter of tasks. The system model is based on *resources* providing services, *tasks* consuming these services, and *event models* specifying task activation patterns. Task activations are triggered by an external source or by events propagated from other tasks (predecessor tasks). The activations in an event model are given by event arrival curves $\eta^-(\Delta t)$ and $\eta^+(\Delta t)$, which return the minimum

and maximum number of events that can arrive in a given time interval Δt, respectively. Their pseudo-inverse counterparts $\delta^+(q)$ and $\delta^-(q)$ return the maximum and minimum time interval between the first and last events in any sequence of q event arrivals, respectively.

The conversion between $\eta^+(\Delta t)$ and $\delta^-(q)$ can be performed as follows [109]:

$$\eta^+(\Delta t) = \max_{q \in \mathbb{N}^+} \{q : \delta^-(q) \leq \Delta t\} \tag{4.1}$$

$$\delta^-(q) = \inf_{\Delta t \in \mathbb{R}_0^+} \{\Delta t : \eta^+(\Delta t) \geq q\} \tag{4.2}$$

The conversion between $\eta^-(\Delta t)$ and $\delta^+(q)$ is omitted but can be similarly derived [135, 109]. It is worth mentioning that event arrival curves are *non-decreasing* and that the upper bound $\eta^+(\Delta t)$ is *sub-additive* [87]. The minimum distance function $\delta^-(q)$ is therefore *non-decreasing* and *super-additive* [109].

The analysis is carried out in a local step and a global loop. In the local step, the local analysis uses the busy window approach [149] to derive each task's response time and output event model. In the global loop, the analysis propagates the output event models of tasks to their dependent tasks, which are activated by the completion of other tasks, becoming then their input event models. The analysis stops when all event models are stable – i.e., a fix point is reached – or when predefined constraints are violated – e.g., the maximum response time or maximum latency of a traffic stream.

4.2.2. Transport and network layer modeling

The analysis problem is modeled in CPA based on the modelling of [13] and illustrated in Figure 4.2. The analysis' starting point is the traffic stream producing packets at the sender according to a given packet-based event model δ_{tx}. The packets are handled at the transport layer by an arbitrary transport protocol. A protocol instance is modeled as a single task τ_{arq} whose response time reflects the protocol behavior. The transport layer then injects traffic into the lower layers of the NoC according to the output event model δ_{arq} of that task.

As illustrated in Figure 4.2, a protocol with handshaking is a bidirectional communication stream. In the NoC, it is mapped as two unidirectional streams: one for the transmitted data (DATA, sender $\rightarrow$ receiver)

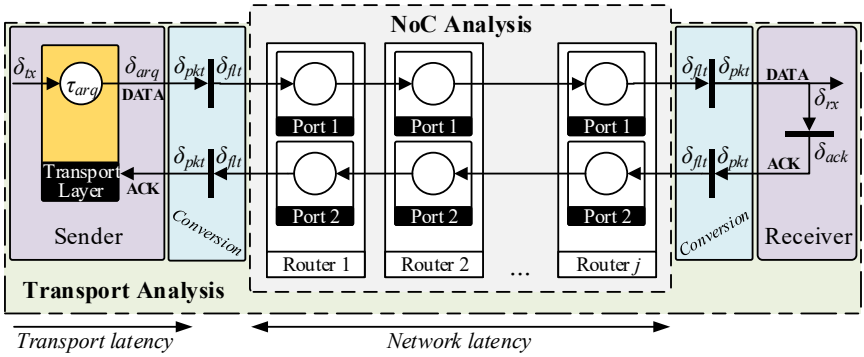


Figure 4.2.: Modeling of transport layer protocol and the underlying NoC in CPA.

and the other for the acknowledgements (ACK, receiver → sender). The lower layers of the NoC, also referred to as the underlying NoC, are modeled in CPA as in [125]: each output port of a router is mapped as a resource, and traffic streams are chains of tasks mapped to resources. The resource arbitration depends on the router arbitration policy. For details on the analysis of the underlying NoC, the interested reader is referred to [125].

The transport layer operates on a packet granularity and so does its analysis – i.e., δ_{tx} and δ_{arq} model packet arrivals. The analysis of the NoC, however, can be performed on a packet or flit granularity depending on whether the network is packet- or wormhole-switched. Thus, an appropriate conversion between the two domains is required. That is shown in Figure 4.2 between the sender and the NoC, and between the NoC and the receiver.

The conversion from a packet-based event model $\delta^-_{pkt,i}(q)$ to a flit-based one $\delta^-_{flt,i}(q)$ is given by:

$$\begin{aligned}\delta^-_{flt,i}(q) = \max\Big\{&(q-1)\cdot d_{min}\,,\\ &\delta^-_{pkt,i}\big(\lceil q \div size_i \rceil\big) + [(q-1) \bmod size_i]\cdot d_{min}\Big\}\end{aligned} \tag{4.3}$$

where the index i refers to a traffic stream i, $size_i$ is the size of any data packet in stream i (in flits), and d_{min} is the minimum distance between two

consecutive flits [125]. The q-th flit activation depends on the activation of the packet containing the q-th flit plus an offset depending on the flit's position in the packet (second term). The first term enforces the minimum distance between q flit activations. For $q \leq 1$, $\delta^-_{flt,i}(q) = 0$.

The conversion from a flit-based event model $\delta^-_{flt,i}(q)$ to a packet-based one $\delta^-_{pkt,i}(q)$ is given by:

$$\delta^-_{pkt,i}(q) = \delta^-_{flt,i}\big((q-1) \cdot size_i + 1\big) \tag{4.4}$$

The q-th packet activation depends on the activation of the first flit of the q-th packet. For $q \leq 1$, $\delta^-_{pkt,i}(q) = 0$.

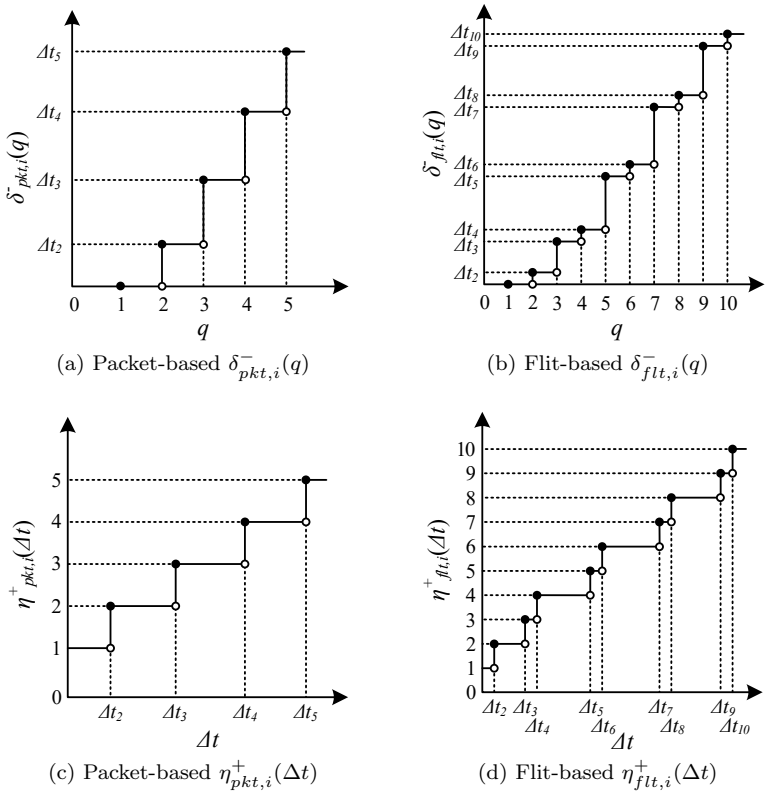
The relation between flit-based and packet-based event models and the relation between the event-arrival and the minimum distance functions of CPA are illustrated in Figure 4.3. Figure 4.3a plots the minimum distance function $\delta^-_{pkt,i}(q)$, which represents packet arrivals. Figure 4.3b plots the respective flit-based function $\delta^-_{flt,i}(q)$ derived from $\delta^-_{pkt,i}(q)$ considering a packet size $size_i = 2$ flits. For every packet activation in $\delta^-_{pkt,i}(q)$, there are two flit activations of $\delta^-_{flt,i}(q)$. The relation between the event-arrival functions and their pseudo-inverse, minimum distance functions, can be seen then between Figures 4.3a and 4.3c and between Figures 4.3b and 4.3d. The conversion between functions is performed with Equations 4.1 and 4.2.

The ACKs are injected back in the network by the receiver δ_{ack}, seen at the right-hand side in Figure 4.2. Protocols acknowledging each packet individually, such as Stop-and-Wait, inject ACKs in the network whenever a packet arrives – i.e., $\delta_{ack} = \delta_{rx}$. Protocols that acknowledge blocks of packets instead must perform an event model conversion from δ_{rx} to δ_{ack}. The acknowledgement event model δ_{ack} is derived from the receiver's input event model δ_{rx} as follows:

$$\delta^-_{ack,i}(q) = \delta^-_{rx,i}\big((q-1) \cdot g_i + 1\big) \tag{4.5}$$

where g_i is the number of packets that the protocol receives before sending an ACK. For $q \leq 1$, $\delta^-_{ack,i}(q) = 0$.

The analysis is then carried out in *two* local steps and a global loop, extending the regular CPA flow with one additional step for the transport layer (cf. Section 4.2.1). First, the NoC analysis computes the worst-case network latency ①. Then, the transport analysis derives the worst-case



(a) Packet-based $\delta^-_{pkt,i}(q)$ (b) Flit-based $\delta^-_{flt,i}(q)$

(c) Packet-based $\eta^+_{pkt,i}(\Delta t)$ (d) Flit-based $\eta^+_{flt,i}(\Delta t)$

Figure 4.3.: Relation between packet-based and event-based event-models (a↔b and c↔d) and between the event-arrival and the minimum distance functions (a↔c and b↔d).

transport latency and updates the output event models accordingly ②. The global loop proceeds as usual in CPA.

The transport layer analysis and derived metrics – e.g., latency – depend, naturally, on the specific protocol that is being modeled. The analysis and metrics of specific protocols are introduced in Section 4.3.

The underlying NoC analysis is arbitrary. That is, the analysis of a two-stage round-robin NoC [125] or the analysis of a NoC with priority-

based arbitration can be used without changing the transport layer analysis. Therefore, the interface between the transport and network analyses is defined next.

4.2.3. Interface with the underlying NoC analysis

The inputs for the NoC analysis are the event models $\delta_{arq,i}$ and $\delta_{ack,i}$, which model the traffic in the data stream i and its ACK stream i_{ack}, respectively. The event model conversion shall be appropriately applied where required, as discussed above.

The output of the underlying NoC analysis is used as input to the transport layer analysis in the form of best- and worst-case Round-Trip Times (RTTs).

Definition 4.1. (Round-Trip Time)
The Round-Trip Time comprises the time it takes for a packet to be transmitted by the sender and received by the receiver plus the time it takes for the respective acknowledgement to be created and transmitted by the receiver and to be received back at the sender.

RTT_i^- and RTT_i^+ are lower and upper bounds on the Round-Trip Time, respectively. The upper bound RTT_i^+ is obtained from the worst-case latency of a data packet L_i^+ and the worst-case latency of its acknowledgement $L_{i_{ack}}^+$ as follows:

$$RTT_i^+ = L_i^+ + L_{i_{ack}}^+ \tag{4.6}$$

Similarly, the lower bound RTT_i^- is calculated using best-case latencies:

$$RTT_i^- = L_i^- + L_{i_{ack}}^- \tag{4.7}$$

The best- and worst-case latencies L_i^- and L_i^+ of any packet in a stream i is then defined as follows:

$$L_i^- = l_i^-(size_i) \tag{4.8}$$

where $l_i^-(size_i)$ is the best-case latency to transmit $size_i$ flits in the NoC; and

$$L_i^+ = l_i^+(size_i) \tag{4.9}$$

where $l_i^+(size_i)$ is the worst-case latency to transmit $size_i$ flits in the NoC. Both $l_i^-(size_i)$ and $l_i^+(size_i)$ are provided by the underlying NoC analysis as seen in [125].

4.3. Formal analysis of ARQ protocols

4.3.1. Stop-and-Wait ARQ

ARQ-based protocols are widely used to guarantee packet delivery [147]. In its simplest variant Stop-and-Wait [147, 90], for each transmitted packet, the sender waits for an ACK from the receiver before transmitting the next packet or, in case of error, retransmitting after a timeout.

An illustrative example is shown in Figure 4.4, where a sender transmits packets one after the other. Stop-and-Wait sends a packet at time ① and waits for an ACK. The receiver acknowledges the successful delivery of the packet at time ②. At time ③, packet 3 is not successfully delivered due to an error in the network. After a timeout ④, the sender retransmits the packet. Out-of-order and duplicated packets are discarded.

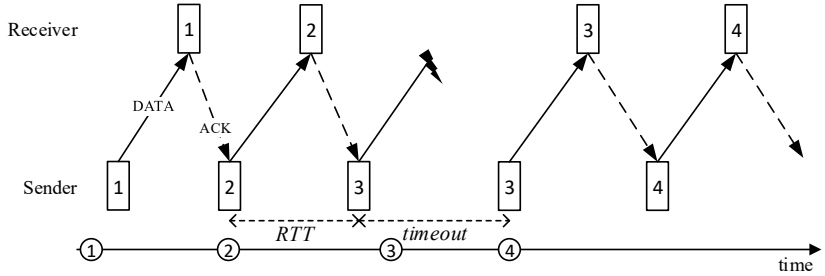


Figure 4.4.: A transmission under Stop-and-Wait ARQ: example 1.

The receiver only accepts and acknowledges packets in-order. However, there are special cases where, due to a lost ACK, the sender retransmits a packet. This is illustrated in Figure 4.5, where, at time ①, packet 3 is successfully received but the ACK is lost due to an error. When the timeout expires ②, the sender retransmits the packet. Since the packet was already accepted by the receiver, at time ③ the packet is discarded but acknowledged nonetheless.

The RTT, also depicted in the figures above, is highly relevant in ARQ-based protocols due to its impact on performance. The timeout of the ARQ must be chosen based on the RTT, so that a retransmission will be triggered only in case of an actual error and not because of latency fluctuations. In Stop-and-Wait ARQ, RTT has also a direct impact on the throughput as

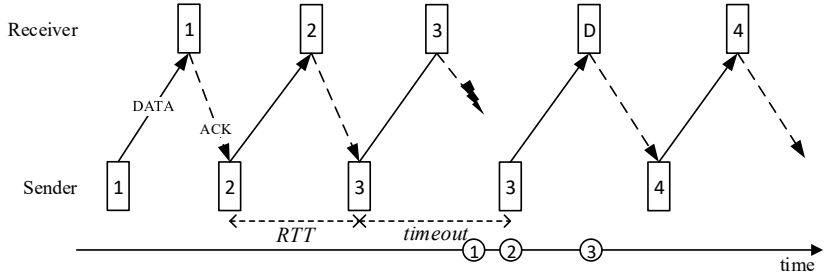


Figure 4.5.: A transmission under Stop-and-Wait ARQ: example 2.

it limits the rate at which a packet can actually be transmitted. In case the RTT becomes a limiting factor, more complex protocols can be used, such as the Go-Back-N.

The timeout is crucial for any handshake-based protocol. It is defined next, for easy reference throughout the chapter.

Definition 4.2. (Timeout)
The timeout period is larger than the worst-case RTT (Definition 4.1).

The Stop-and-Wait protocol can be seen as a special case of the Go-Back-N protocol, which is introduced in the sequel. The interested reader can refer to Appendix A for more details on Stop-and-Wait ARQ.

4.3.2. Go-Back-N ARQ

The limited throughput of Stop-and-Wait is improved in Go-Back-N [147]. Go-Back-N allows n packets to be sent before stopping and waiting for an acknowledgement, thus, allowing the transmission of bursts of packets with periods shorter than the RTT. To achieve that, it introduces the concept of a *send window* with a parameterizable size, usually denoted n.

An illustrative example is shown in Figure 4.6. Go-Back-N sends $n = 3$ packets at time ① before waiting for an ACK. The receiver acknowledges the successful delivery of each packet ②. In ③, packet 7 is not successfully delivered. After a timeout t_{out}, the sender retransmits all unacknowledged packets ④. Out-of-order and duplicated packets are discarded. Notice that Stop-and-Wait can be seen as a case of Go-Back-N with a send window of size $n = 1$.

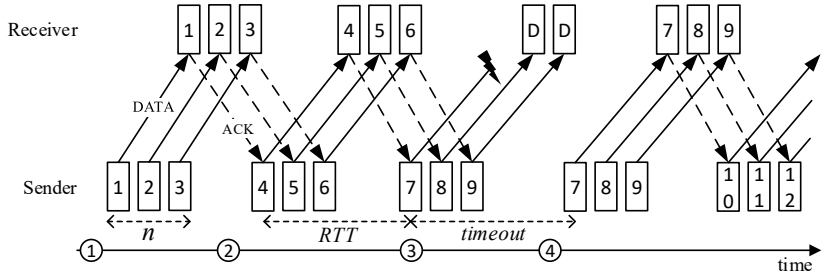


Figure 4.6.: A transmission under Go-Back-N ARQ ($n=3$): example 1.

A similar situation as seen in Stop-and-Wait (Figure 4.5) occurs in Go-Back-N. Although the receiver only accepts packets in-order, there are special cases where, due to a lost ACK, the receiver will acknowledge the same packet twice. This is illustrated in Figure 4.7, where, at time ①, packets 7, 8 and 9 are successfully received but an ACK is lost due to an error. Upon the timeout expiration, at time ②, the sender retransmits, starting with the earliest unacknowledged packet. Since packets 7, 8 and 9 were already accepted by the receiver, at time ③ the packets are discarded but acknowledged nonetheless.

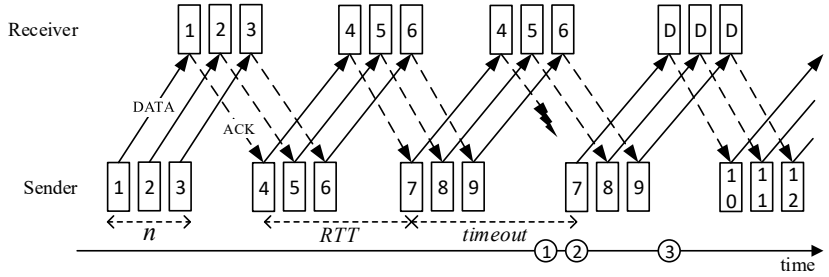


Figure 4.7.: A transmission under Go-Back-N ARQ ($n=3$): example 2.

Thanks to the modelling proposed in this thesis, the transport layer analysis for NoCs can reuse existing analysis for general packet-switched networks. For the sake of completeness and discussion of experimental results later on, relevant parts of analysis of Go-Back-N ARQ of Axer are reproduced in Appendix A along with more details about the protocol.

4.3.3. DMA ARQ

Go-Back-N ARQ performs well for traffic without or with very small bursts. With bursts, as in the case of DMA transfers, it requires large send windows to deliver good performance [147], implying large retransmission buffers and, thus, area and power overheads [104]. To achieve lower latencies with reduced area overhead, an ARQ protocol optimized for DMAs, the DMA ARQ, is proposed. The protocol is a variation of Go-Back-N, where the send window is increased to match the transfer size. In case of errors, packets are selectively repeated, similar to Selective Repeat [147]. The protocol's implementation is enabled by exploiting two properties found in DMA transfers.

First, it exploits the fact that the number of packets in a DMA transfer, the transfer size n_{dma}, is known beforehand. That allows the protocol to acknowledge the whole transfer instead of each packet individually. As a result, the throughput of a transfer is not limited by the protocol. In case the transfer was not successful, a NACK is sent, selectively requesting the retransmission of missing and corrupt packets. Correct packets can be processed and have their contents forwarded to the memory controller as soon as they arrive, since the memory address of each packet's contents can be inferred from the transfer's initial address.

Second, due to the nature of DMA transfers, it is not necessary to keep packets in a retransmission buffer. In case of retransmission, the data from missing or corrupt packets can be read from the sender's local memory. Notice that this is only possible for DMA traffic. In general traffic, the data is not accessible – e.g., a non-blocking uncached write issued by a processor. The protocol implementation avoids overhead in area but increases the overhead in time in case of a retransmission, requiring an additional time t_{mem} to read from the local memory. Furthermore, the protocol assumes that memory consistency is guaranteed independently by a mechanism in software or hardware, as seen in modern systems.

An illustrative example of an error free scenario is shown in Figure 4.8. The sender starts a transfer with size $n_{dma} = 9$ at time ①. All packets are immediately transmitted back-to-back. At the end of the transfer ②, the receiver acknowledges the entire transfer. After receiving the acknowledgement that finishes the transaction ③, the protocol is ready for a second transfer, which starts after some time ④.

A second example illustrates in Figure 4.9 the protocol operating un-

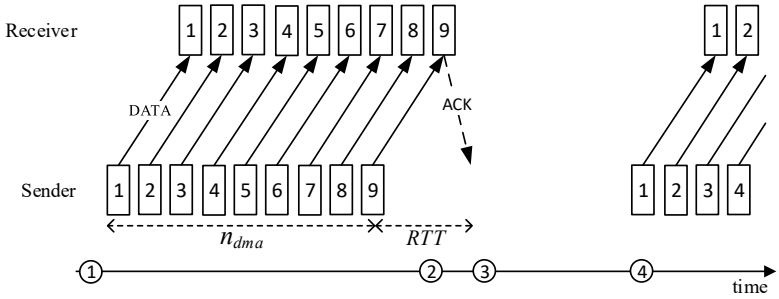


Figure 4.8.: Transmission under DMA ARQ: example 1.

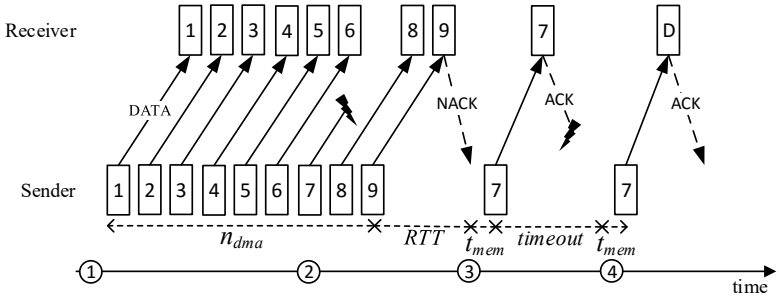


Figure 4.9.: Transmission under DMA ARQ: example 2.

der the occurrence of two errors. The sender starts a transfer with size $n_{dma} = 9$ at time ①. All packets are immediately transmitted back-to-back. Eventually, a packet is not successfully delivered ②, and the receiver sends a NACK at the end of the transfer. The data to be retransmitted, which was transported in packet 7, is read from the local memory and the packet is retransmitted at time ③. Packet 7 is received but the acknowledgement is lost. After a timeout ④, the sender retransmits the last packet of the transfer. The duplicate packet is dropped but still acknowledged, and the transfer ends.

The interested reader can refer to Appendix A for more details on DMA ARQ, including its finite state machine.

The worst-case communication time analysis of DMA ARQ is presented in two parts. First, it addresses the protocol behavior in the error-free sce-

nario. Then, the worst-case impact of errors on a transmission is evaluated and incorporated into the analysis. The analysis models the data transfer itself, corresponding to a write DMA transfer. A write differs from a read in that the latter has a request packet that starts the transfer. The analysis assumes that the network interface itself is not faulty and that it is able to detect packet corruption – e.g., using CRC-based integrity check.

4.3.3.1. Formal analysis: the error-free case

To obtain the worst-case end-to-end latency of a packet under DMA ARQ, it is necessary to derive, among others, the contribution of the protocol to the latency. That is captured by the WCRT of DMA ARQ $R^+_{dma,i}$, which is the largest period of time in which a packet is retained by the protocol. The analysis relies on the busy window approach [149]. Therefore, the analysis starts by deriving the *busy period* for the protocol.

The busy period $w_{dma,i}$ is the largest time interval in which packets arrive at the transport layer and need to be queued while the DMA ARQ protocol is waiting for ACKs of previous packets. It is given by:

$$w_{dma,i} = \left\lfloor \frac{\eta^+_{tx,i}(w_{dma,i}) - 1}{n_{dma}} \right\rfloor \cdot RTT^+_i \tag{4.10}$$

Equation 4.10 forms an integer fixed point problem typical in busy-window-based analyses – $w_{dma,i}$ is present on both sides of the equation. The problem can be solved iteratively, starting with a very small $\epsilon > 0$ ($w_{dma,i} = \epsilon$).

Lemma 4.1. *The busy window is upper bounded by Equation 4.10.*

Proof. The proof is by direct deduction. In DMA ARQ, all packets are transmitted as soon as they are generated. The only blocking that can be caused by the protocol to a packet is when a transfer ends. At the end of each transfer, the protocol waits for an acknowledgement before starting the next one, taking at most RTT^+_i per complete transfer of n_{dma} packets. That is exactly what is captured by $w_{dma,i}$ in Equation 4.10, which is therefore an upper bound. □

The Worst-Case Multiple Packet Forwarding Time $B^+_{dma,i}(q)$ is the largest time interval to forward a sequence of q packets, here assuming error-free

conditions. Considering that the first packet of a sequence is starting a transfer, the $B^{+}_{dma,i}(q)$ of a stream i is given by:

$$B^{+}_{dma,i}(q) = \left\lfloor \frac{q-1}{n_{dma}} \right\rfloor \cdot RTT^{+}_{i} + \delta^{-}_{tx,i}(q) \tag{4.11}$$

Lemma 4.2. *The Worst-Case Multiple Packet Forwarding Time $B^{+}_{dma,i}(q)$ given by Equation 4.11 is an upper bound.*

Proof. The proof is by induction. When $q = 1$, the packet can be forwarded immediately, since it is starting a transfer and the protocol is idle. In a subsequent $q+1$-th activation, the packet can be immediately forwarded unless it belongs to a subsequent transfer, i.e., it is the first packet $q = x \cdot n_{dma} + 1$ of a subsequent transfer $x > 1$. In that case, Equation 4.11 must account for the delay due to waiting for an ACK, which takes at most time RTT^{+}_{i}, every subsequent new transfer, i.e., every n_{dma} packets. Additionally, the packet cannot be forwarded before its arrival $\delta^{-}_{tx,i}(q)$. That results in Equation 4.11. □

The Best-Case Multiple Packet Forwarding Time $B^{-}_{dma,i}(q)$ is the smallest time interval to forward q packets. Considering that the first packet is starting the transfer, it is given by:

$$B^{-}_{dma,i}(q) = \left\lfloor \frac{q-1}{n_{dma}} \right\rfloor \cdot RTT^{-}_{i} \tag{4.12}$$

Lemma 4.3. *The Best-Case Multiple Packet Forwarding Time $B^{-}_{dma,i}(q)$ given by Equation 4.12 is a lower bound.*

Proof. The proof is omitted for brevity, since it follows the same reasoning as the proof of Lemma 4.2, but considering the best-case round-trip latency RTT^{-}_{i} instead of RTT^{+}_{i}. □

The Worst-Case Response Time $R^{+}_{dma,i}$ is the largest time interval that any packet is delayed by DMA ARQ before being forwarded to the network. The $R^{+}_{dma,i}$ of a stream i is bounded by:

$$R^{+}_{dma,i} = \max_{1 \le q \le \eta^{+}_{tx,i}(w_{dma,i})} \{B^{+}_{dma,i}(q) - \delta^{-}_{tx,i}(q)\} \tag{4.13}$$

Theorem 4.1. *$R^+_{dma,i}$ (Equation 4.13) provides an upper bound on the response time of an arbitrary packet in the traffic stream i transmitted under DMA ARQ.*

Proof. The WCRT of an arbitrary packet in the traffic stream i is obtained with the busy window approach [149]. The response time of the q-th packet is the time between its arrival ($\delta^-_{tx,i}(q)$, a lower bound) and its injection in the network ($B^+_{dma,i}(q)$, an upper bound). The WCRT is then found as the maximum among the response times of activations occurring inside the busy window $w_{dma,i}$ [149]. It remains to prove that the busy window is correctly captured by Equation 4.10, and that the blocking captured in Equation 4.11 is an upper bound. Those are proved in Lemmas 4.1 and 4.2, respectively. □

The event model capturing the traffic injection in the network by DMA ARQ can now be derived. The output event model $\delta^-_{dma,tx}$ propagated by the transport layer, under error-free conditions, is obtained as follows:

$$\delta^-_{dma,tx}(q) = \max\{\delta^-_{tx}(q) - R^+_{dma} + R^-_{dma}, B^-_{dma}(q-1)\} \tag{4.14}$$

Theorem 4.2. *The minimum distance function $\delta^-_{dma,tx}(q)$ given by Equation 4.14 is a lower bound.*

Proof. Packets can leave DMA ARQ as soon as they arrive and are processed but not faster than the DMA ARQ is able to process them. That is captured by the max function. The proof is by cases, with two cases that must be lower bounds. The first case is when the packets leave the DMA ARQ as fast as they are arrive. Since packets can be affected by delay in the DMA ARQ resulting in a jitter ($R^+_{dma} - R^-_{dma}$) that is propagated with the output event model. That is guaranteed to be a lower bound. The proof is given in [137]. The second case is that any q packets cannot be closer to each other than what the DMA ARQ is able to process. That is captured by $B^-_{dma}(q-1)$, which is proved to be a lower bound in Lemma 4.3. Since both cases are lower bounds, Equation 4.14 is also a lower bound. □

Finally, the time it takes to complete a transfer using DMA ARQ can be bounded. The *overall latency* $\mathbb{L}^+_{dma,i}(q)$ of a transfer comprising q data packets in a stream i is given by:

$$\mathbb{L}^+_{dma,i}(q) = \delta^-_{tx,i}(q) + L^+_i + R^+_{dma,i} \tag{4.15}$$

where one complete transfer consists of $q = n_{dma}$ packets.

Theorem 4.3. *Equation 4.15 gives an upper bound on the overall latency to complete a transfer of q data packets.*

Proof. The proof is by direct deduction. The latency consists of the time it takes for the sender to create q packets ($\delta^{-}_{tx,i}(q)$), plus the latency it takes for the last (q-th) packet to be delivered by the network, plus the worst-case delay for that packet introduced by the protocol ($R^{+}_{dma,i}$). Due to causality – i.e., packets cannot bypass each other – all previous packets must have been received by the time the last packet is received. Thus, Equation 4.15 is a valid upper bound. □

4.3.3.2. Formal analysis: the error case

In this section, the error-free assumption is removed and the analysis is extended to consider scenarios where a number of errors affect the transmission, the so-called k-error scenario [8, 13]. A k-error scenario is a scenario where during the transmission of q packets, k error occurrences are assumed. The formal analysis conservatively assumes that each error occurrence has the worst-case impact.

As described in Chapter 2, errors can have several types of effects in the NoC, such as latency increase, packet corruption, and packet loss. This analysis assumes that the wost case impact of an error is packet loss. Moreover, the analysis assumes that errors are transient. Those assumptions have the following consequences. Packet corruption can be seen as a case of packet loss – corruption causes the packet to be eventually discarded by the receiver after an integrity check. An error-induced latency increase is assumed to be lower than and bounded by the time it takes to retransmit a lost packet. In case the assumptions above do not hold during operation – for example due to a permanent fault or a transient fault with static effects –, their violation can be detected and reported by monitoring the protocol's performance during runtime. That is, if an application is dimensioned to allow a maximum of $k = 2$ errors in a given transmission, the occurrence of $k > 2$ error occurrences, which could lead to timing violations, can be monitored and reported to the system controller or OS as hardware failure.

The worst-case impact of a packet loss on a transfer under the DMA ARQ protocol is when the ACK packet is lost right before being delivered,

similar to Go-Back-N (cf. Section A.2.2 and [13]). It causes the maximum load in the network and the maximum delay. The intuition is given by means of an example in the sequel, followed by a proof in Lemma 4.4.

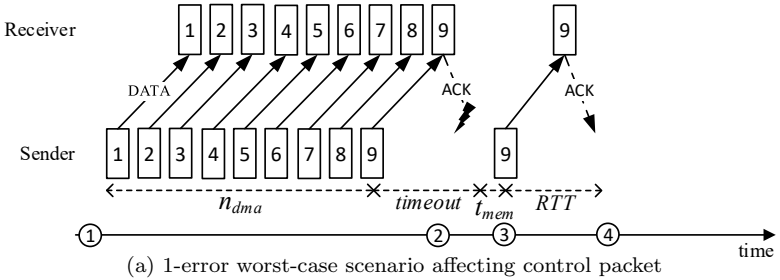
Figure 4.10 illustrates the three possible candidates for a 1-error scenario. The error can either affect a control packet (ACK) (cf. Figure 4.10a), a data packet (cf. Figure 4.10b), or a trailing data packet (cf. Figure 4.10c). A 2-error scenario affecting control packets is shown in Figure 4.10d.

The *1-error worst-case scenario* is shown in Figure 4.10a. After ① all $n_{dma} = 9$ data packets of the transfer arrive at the receiver, the ACK packet is lost ②. That causes ③ a retransmission of the last data packet (local memory access t_{mem} plus RTT_i^+), which is triggered after a timeout t_{out}. The ACK is then received and the transfer ends successfully at time ④. In the worst case, $t_{out} + t_{mem} + RTT_i^+$ additional time is required. In a *2-error scenario* (Figure 4.10d), a second retransmission occurs after a timeout ④ due to a second error, after which the transfer ends successfully ⑤. In summary, in the worst case, $k \cdot (t_{out} + t_{mem} + RTT_i^+)$ additional time is required for k errors.

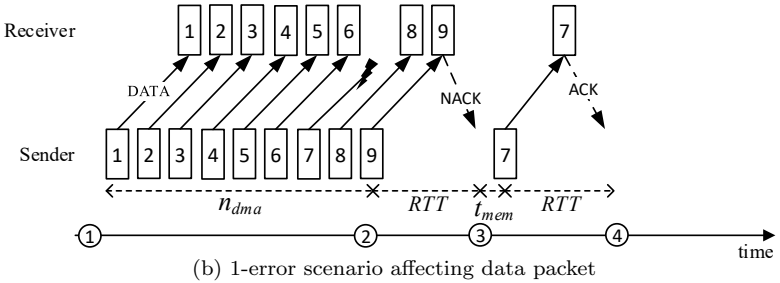
For the sake of completeness, Figure 4.10b illustrates an alternative 1-error scenario where the error affects a data packet ②. In that case, at the end of the transfer ③ the receiver sends a NACK requesting the retransmission of the missing data packet. That requires $t_{mem} + 2 \cdot RTT_i^+$ additional time, which is, by definition, smaller than $t_{out} + t_{mem} + RTT_i^+$ (cf. Definition 4.2). The third possible 1-error scenario is shown in Figure 4.10c, where the error affects the trailing data packet ②. In that case, the receives no ACK and retransmits the trailing data packet after a timeout ③. That requires $t_{out} + t_{mem} + RTT_i^+$ additional time, similar to the aforementioned worst case. Notice, however, that the scenario injects less traffic in the network than the aforementioned worst case, as one less ACK packet is transmitted. The general k-error worst-case scenario is featured in Lemma 4.4, including a proof.

Lemma 4.4. *The worst-case impact of a packet loss on a transfer is when the ACK packet is lost right before being delivered (cf. Figures 4.10a and 4.10d). It assumes that the loss of any data packet does not cause the failure of the entire transmission. In the worst case,* $k \cdot (t_{out} + t_{mem} + RTT_i^+)$ *additional time is required for* k *errors.*

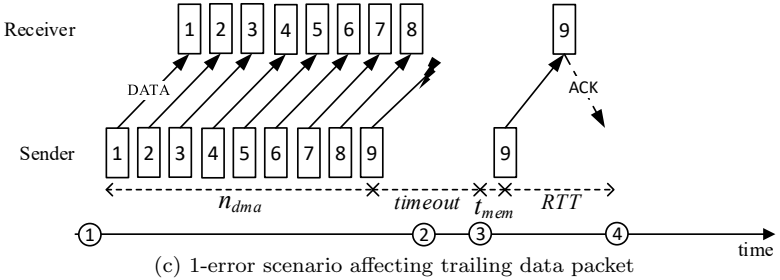
Proof. The proof is by direct deduction.



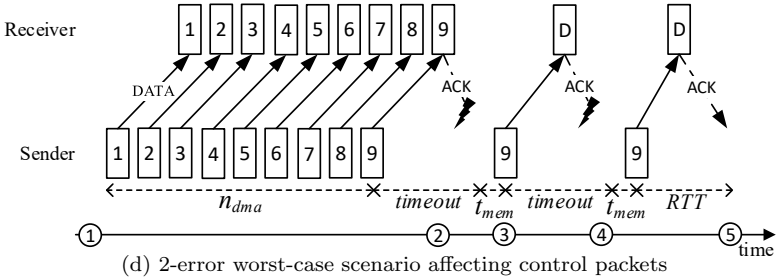
(a) 1-error worst-case scenario affecting control packet



(b) 1-error scenario affecting data packet



(c) 1-error scenario affecting trailing data packet



(d) 2-error worst-case scenario affecting control packets

Figure 4.10.: Transmission under DMA ARQ: error scenarios.

Table 4.1.: Different ways errors can affect a DMA ARQ transmission.

Affected packet	**Consequence**	**Additional time**
Data packet (first)	retr. + mem.	$t_{mem}+RTT_i^+$
Data packet (subseq.)	retr. + mem.	t_{mem}
Trailing data packet	timeout + retr. + mem.	$t_{out}+t_{mem}+RTT_i^+$
Retrans. data packet (first)	timeout + retr. + mem.	$t_{out}+t_{mem}+RTT_i^+$
Retrans. data packet (subseq.)	retr. + mem.	t_{mem}
Trailing retrans. data packet	timeout + retr. + mem.	$t_{out}+t_{mem}+RTT_i^+$
Control packet (ACK/NACK)	timeout + retr. + mem.	$t_{out}+t_{mem}+RTT_i^+$

The possible ways an error can affect a DMA ARQ transmission is summarized in Table 4.1, which shows what packet was affected by the error, what is the consequence in the protocol execution (timeout, retransmission, and memory access) and what is the additional time required. Notice that some scenarios are only possible in case of multiple errors – e.g., an error affecting a retransmitted data packet – those are emphasized in *italic*. Moreover, the table differentiates between a first occurrence of an error and subsequent ones, which present different timing impact – e.g., retransmitting a second data packet in a same burst requires less additional time since it reuses the acknowledgement already required for the first retransmitted data packet.

From all possible error impacts, an error (1-error scenario) affecting either a trailing data packet, a retransmitted data packet, a trailing retransmitted packet or a control packet present the most drastic consequence: a timeout event triggers a retransmission requiring a memory access. That corresponds to $t_{out} + t_{mem} + RTT_i^+$ additional time in the transmission that was directly affected by the error.

When considering a k-error scenario, the worst-case impact is when each of the k error occurrences result in $t_{out} + t_{mem} + RTT_i^+$ additional time. Thus, the k-error scenario results in $k{\cdot}(t_{out}+t_{mem}+RTT_i^+)$ total additional time.

Finally, since the choice of the aforementioned scenarios is arbitrary in terms of protocol execution, one scenario is chosen to represent the worst-case scenario. Due to its simplicity and due to the fact that it results in the

largest number of packets[1] injected in the NoC, the scenario where every error in a k-error scenario affects a control packet (ACK) is chosen as the worst-case scenario. Losing the ACK packet causes the retransmission of the last packet is triggered after a timeout t_{out} (cf. Figure 4.10a at time ③). The retransmission requires a local memory access with time t_{mem} plus time RTT_i^+ for sending a data packet and receiving its acknowledgement. Similarly, a second error causes a second retransmission after a timeout (cf. Figure 4.10d at time ④). The resulting additional time is $k \cdot (t_{out} + t_{mem} + RTT_i^+)$, for k errors. □

First, the impact of errors is integrated into the busy period. The k-error busy period $w_{dma,i}(k)$ is now given by:

$$w_{dma,i}(k) = k \cdot (t_{err} + RTT_i^+) + \left\lfloor \frac{\eta_{tx,i}^+(w_{dma,i}(k)) - 1}{n_{dma}} \right\rfloor \cdot RTT_i^+ \qquad (4.16)$$

where $t_{err} = t_{out} + t_{mem}$. In the worst case, in addition to the error-free busy window, each error leads to a timeout t_{out} and a retransmission RTT_i^+ (data packet and ACK).

Lemma 4.5. *The k-error busy period $w_{dma,i}(k)$ given by Equation 4.16 is an upper bound.*

Proof. The proof is by direct deduction. Equation 4.10 presents the busy window in the error-free case and Lemma 4.1 shows that it is an upper bound. Lemma 4.4 shows that the worst-case impact of k errors is $k \cdot (t_{out} + t_{mem} + RTT_i^+)$. Equation 4.16 extends the busy window of Equation 4.10 (error-free) with the worst-case impact of k errors. Thus, Equation 4.16 is also an upper bound. □

Similarly, the multiple packet forwarding time under errors is derived, accounting for the impact of k errors on the transfer. The k-error Worst-Case Multiple Packet Forwarding Time $B_{dma,i}^+(q,k)$ is given by:

$$B_{dma,i}^+(q,k) = k \cdot (t_{err} + RTT_i^+) + \left\lfloor \frac{q-1}{n_{dma}} \right\rfloor \cdot RTT_i^+ + \delta_{tx,i}^-(q) \qquad (4.17)$$

[1] See Figures 4.10a, 4.10b and 4.10c for the 1-error scenario. A proof for this argument is not given since it is not essential for the proof of Lemma 4.4.

In addition to $B^+_{dma,i}$, in the worst case, each error leads to a timeout and a retransmission whose impact is also bounded by $t_{out} + RTT^+_i$, as in Equation 4.16.

Lemma 4.6. *The k-error Worst-Case Multiple Packet Forwarding Time $B^+_{dma,i}(q,k)$ given by Equation 4.17 is an upper bound.*

Proof. The proof is by direct deduction. Equation 4.11 presents the busy window in the error-free case and Lemma 4.2 shows that it is an upper bound. Lemma 4.4 shows that the worst-case impact of k errors is $k \cdot (t_{out} + t_{mem} + RTT^+_i)$. Equation 4.17 extends Equation 4.11 (error-free) with the worst-case impact of k errors. Thus, Equation 4.17 is also an upper bound. □

The k-error Worst-Case Response Time $R^+_{dma,i}(q)$ is derived from Eq. 4.13 to account for the k-error versions of the functions $w_{dma,i}(k)$ and $B^+_{dma,i}(q,k)$:

$$R^+_{dma,i}(k) = \max_{1 \le q \le \eta^+_{tx,i}(w_{dma,i}(k))} \{B^+_{dma,i}(q,k) - \delta^-_{tx,i}(q)\} \tag{4.18}$$

Theorem 4.4. *$R^+_{dma,i}(k)$ (Equation 4.18) provides an upper bound on the response time of an arbitrary packet in the traffic stream i transmitted under DMA ARQ and a k-error scenario.*

Proof. The proof is analogous to the error-free version in Theorem 4.1, but considering the k-error variant of the functions. □

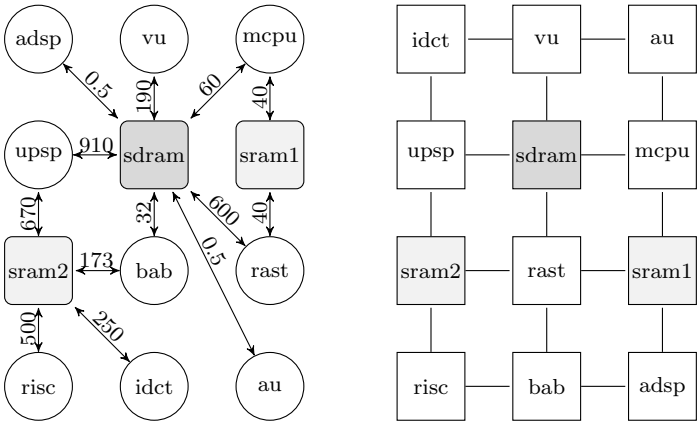
Finally, the time it takes to complete a transfer using DMA ARQ under errors is bounded. The *k-error overall latency* $\mathbb{L}^+_{dma,i}(q,k)$ of a transfer comprising q data packets in a stream i is given by:

$$\mathbb{L}^+_{dma,i}(q,k) = \delta^-_{tx,i}(q) + L^+_i + R^+_{dma,i}(k) \tag{4.19}$$

Besides the latency in the error-free case (Equation 4.15), in the k-error scenario, Equation 4.19 must account for the latency overhead. This is included by the last term, the k-error worst-case response time $R^+_{dma,i}(k)$.

Theorem 4.5. *Equation 4.19 gives an upper bound on the overall latency to complete a transfer of q data packets under a k-error scenario.*

Proof. The proof is analogous to the error-free version in Theorem 4.3, but considering the k-error variant of the functions. □



(a) MPEG decoder. (b) Mapping on 3x4 NoC.

Figure 4.11.: The node graph annotated with communication bandwidth requirements (MB/s), and mapping of the MPEG decoder application.

4.4. Experimental evaluation

The experiments evaluate in a real-time context the performance of memory traffic of the MPEG decoder (Figure 4.11) on a NoC implementing end-to-end Stop-and-Wait, Go-Back-N, and DMA ARQ. First, the performance of the general memory traffic analyzed and compared with simulation results. Then, the performance of the memory traffic is evaluated when parts of it consist of DMA transfers. The analyses were implemented in PyCPA [34] and the simulations were carried out in the OMNeT++ network simulator [154].

The employed NoC consists of 3x4 nodes in a 2D-mesh topology. Each node consists of a tile, a router, and a network interface connecting both. Links connect the routers and network interfaces. The NoC implements wormhole-switching, virtual channel flow control, XY source-routing and SLIP arbitration in the routers. The frequency of operation is 800MHz. The underlying NoC analysis [125] is utilized with overhead parameters per hop O_r and per path O_p set to 0.

The MPEG decoder [152] employed in the experiments is presented in Figure 4.11a, which shows the node graph and the bandwidth requirements (MB/s) between the communicating nodes. In the experiments, the data flows from the memories to the nodes. The nodes have been mapped to a 3x4 NoC as depicted in Figure 4.11b [105]. Each arrow represents the source and destination of a *traffic stream*. A traffic stream referred to as *node.X*, where *node* is the processing element and X is the memory: 1 for *sram1*, 2 for *sram2*, and S for *sdram*. For instance, *rast* has two streams: *rast.1* and *rast.S*.

4.4.1. General traffic

The first part of the experiments evaluates the performance of traffic streams consisting of periodic, non-bursty accesses to memory. It is assumed that the traffic in real-time systems exhibits well understood and predictable patterns – e.g., streaming applications [152] or predictable execution models such as superblocks [138, 117]. Therefore, the size of the data accesses of each MPEG node is varied in three different scenarios: 64, 128 and 256Bytes transported in a packet with 5, 9 and 17 flits, respectively. The frequency of the data accesses is derived to match the specified bandwidth – e.g., 500 MB/s (*risc.2*) corresponds to 64B every 128ns (102 cycles at 800MHz).

Figures 4.12a, b and c show the maximum observed and analytical worst-case latencies for a packet in a given traffic stream for increasing access sizes. For Stop-and-Wait, analysis (SNW) provides valid bounds to simulation (SimSNW) for all accesses sizes, as it is to be expected. The margin over simulation is noticeably higher for traffic to/from the *sdram*. That comes from the underlying NoC analysis: all traffic streams coexist in the NoC in that setup and share the same network interface to access the *sdram*, constituting a bottleneck. Moreover, the values from simulation are observed maxima. The true worst-case lies somewhere between the values from the analysis and the observed ones.

Switching from Stop-and-Wait to Go-Back-N $n = 2$ (GBN2) and further increasing the send window ($n > 2$, not shown in the figures) does not decrease latency. This type of traffic is characterized by the *quasi* absence of bursts. As a result, small retransmission buffers can be employed for general traffic without compromising performance. Not only that, but if the distance between two packets being injected in the network is smaller

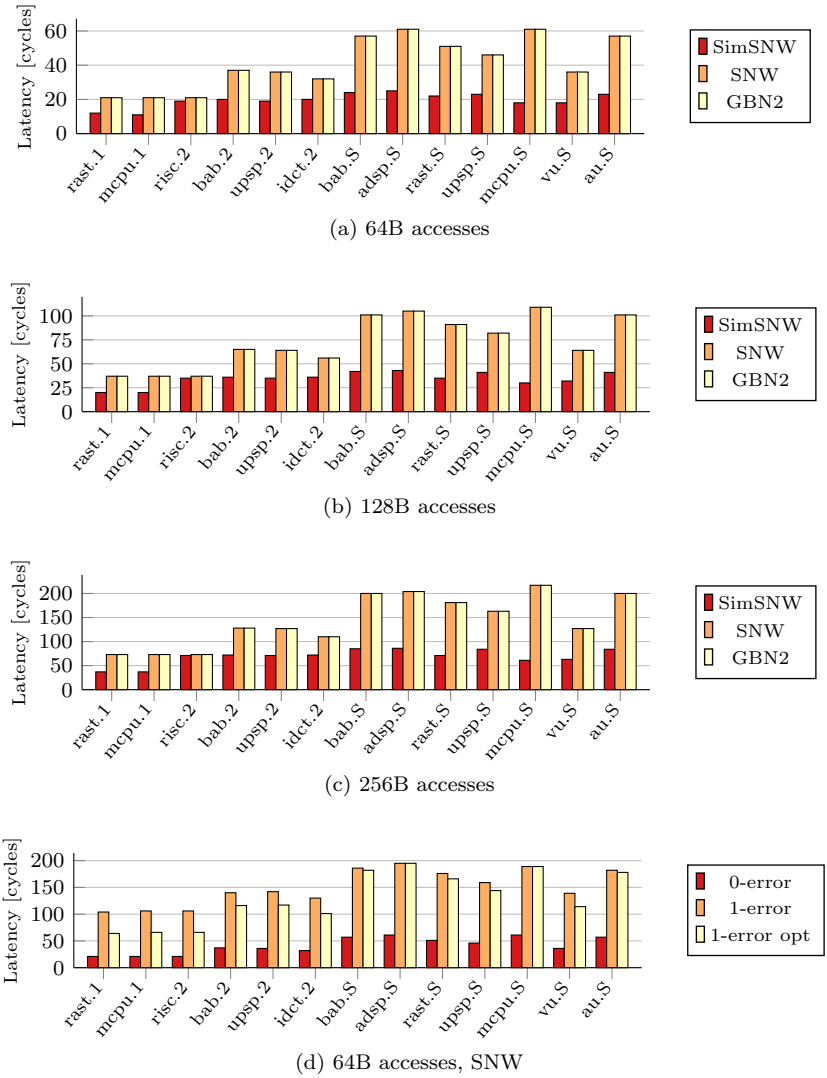


Figure 4.12.: Maximum end-to-end packet latencies for Stop-and-Wait (observed in simulation SimSNW, and analytical SNW) and Go-Back-N with $n = 2$ (analytical GBN2). Error-free scenarios in (a,b,c), error scenario in (d).

than the RTT, Go-Back-N has no gain over Stop-and-Wait, and a single-slot retransmission buffer can be employed. However, that decision can critically affect the performance of bursty traffic, such as DMA transfers.

The impact of errors on transmissions is shown in Figure 4.12d for 64B accesses and Stop-and-Wait. The figure shows the worst-case latency when employing a safe, equal timeout t_{out} for all streams (1-error), and when optimizing the timeout individually per stream (1-error opt), which is slightly larger than its latency bound. Since the worst-case retransmission is largely dominated by the timeout, optimizing it results in up to 38% lower latencies in a 1-error scenario.

4.4.2. DMA traffic

The second part of the experiments analyzes the performance of bursty memory traffic, which is characteristic of DMA transfers. That type of traffic is predominant in safety-critical MPSoCs platforms [138]. Therefore, it is assumed that nodes with high throughput to the *sdram* use DMA to be more efficient – i.e., streams *rast.S*, *upsp.S*, and *vu.S*. Three scenarios are evaluated where transfers have 4KB, 8KB and 16KB. The frequency with which the transfers occur are set to match the specified bandwidth. Each transfer consists of 32, 64, and 128 9-flit-long packets, respectively for each scenario. It is assumed that the DMA transfers are synchronized in software or hardware, such that one transfer is performed at a time. That reflects real hardware limitation and programming model. The non-DMA streams[2] are configured as 64B, Stop-and-Wait general traffic.

Figures 4.13a, b and c show the maximum latency bounds for entire DMA transfers in the error-free case. For transfers of same size (e.g., 8KB) maximum latency varies more than 300%. The extra delay is caused by background traffic, specially by the stream from *upsp.S*, and is captured by the underlying NoC analysis. Stop-and-Wait is not suitable for bursty transfers, resulting in very high latencies and in the unschedulability of *upsp.S* with 16KB transfers (Figure 4.13c). Go-Back-N's performance improves as its send window is increased. Nonetheless, DMA ARQ presents the lowest latency bound in all cases. Only for *vu.S*, due to very low network interference, Go-Back-N is able to achieve the same performance at the expense of a large retransmission buffer (send window 4). Despite the

[2] All other streams already evaluated in Section 4.4.1 except *rast.S*, *upsp.S*, and *vu.S*, which are here configured as DMA traffic.

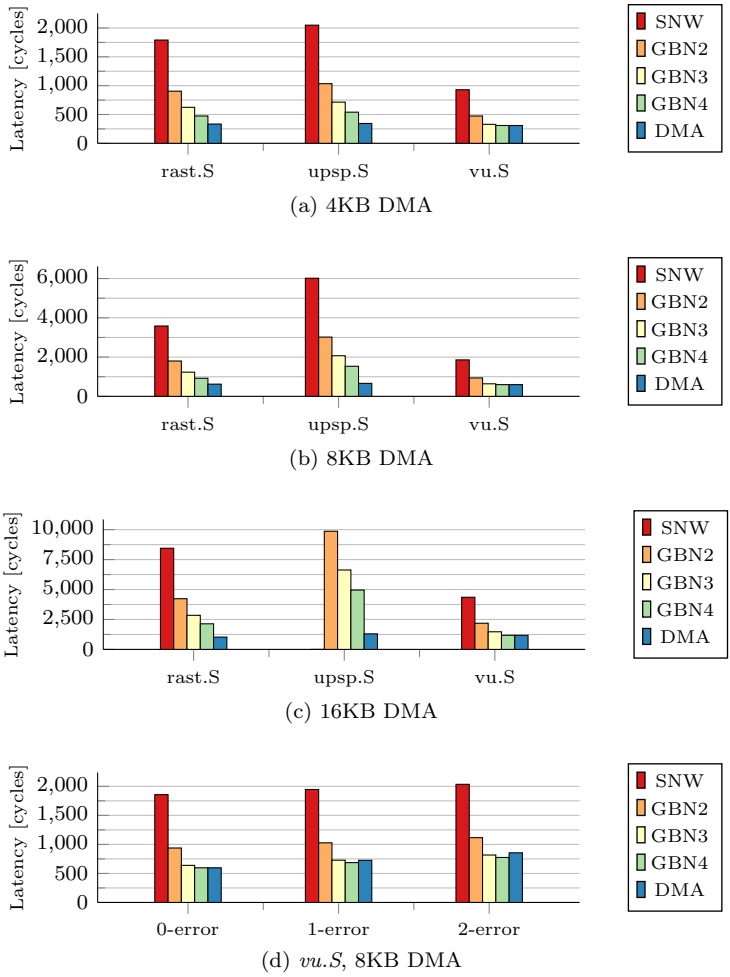


Figure 4.13.: Worst-case end-to-end latency of DMA transfers for Stop-and-Wait (SNW) and Go-Back-N with $n = 2$, 3 and 4 (GBN2, GBN3, GBN4) and DMA ARQ (DMA). Error-free scenarios in (a,b,c), error scenarios in (d).

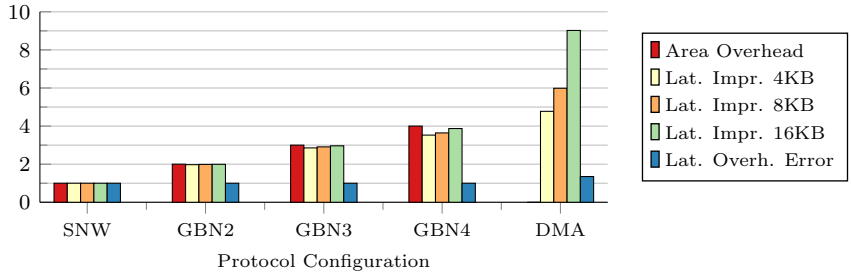


Figure 4.14.: Comparison of the performance of ARQ-based protocols, normalized to Stop-and-Wait.

latency variations for the DMAs, the latency bounds for the non-DMA streams (not shown in the figure) remain the same when varying the protocols.

When considering error scenarios, DMA ARQ outperforms all protocol configurations except for stream *vu.S*, detailed in Figure 4.13d. That is explained by the fact that DMA ARQ requires extra time to read from the local memory when retransmitting. Since in the error-free case (0-error) GBN3, GBN4 and DMA deliver similar latency bounds, DMA presents slightly higher latencies in error scenarios (5% higher than GBN4 in 1-error). The analysis considers $t_{out} = 60$ and $t_{mem} = 40$ cycles.

4.4.3. Comparison

The third part of the experiments evaluates the impact of the different transport protocols on the system. Figure 4.14 shows the latency improvement, area overhead, and error overhead when varying the protocol configuration. The values are normalized to a 3x4 NoC with Stop-and-Wait. Area overhead metric covers retransmission buffer requirements.

The results show that extending the send window in Go-Back-N introduces more area overhead than it increases performance, specially for smaller transfer sizes. Notice that the reported area overhead scales with the system – i.e., the values in the Figure are multiplied by the number of nodes in the NoC. Notice also that, for DMA and general traffic to transmit in parallel, a node would require at least two protocol instances.

Error overhead measures the additional time required to finish the transfer in a *1-error* scenario. In the worst-case, it takes Go-Back-N the same time as Stop-and-Wait. DMA ARQ, on the other hand, presents a major performance improvement (6.6x on average) while avoiding additional buffers. That comes at the price of additional time overhead when recovering from an error, since the data must be fetched from the local memory instead of a retransmission buffer. That penalty, however, is only seen during error occurrences and does not increase with transfer length.

4.5. Summary

This chapter addressed the use of ARQ-based protocols in real-time NoCs. An analysis framework was introduced by integrating existing formal analyses of the transport layer and analyses of the network layer. The framework can be used to provide formal latency guarantees for on-chip traffic in real-time systems and it can be easily composed with different protocols and NoC models. Moreover, the chapter also introduced DMA ARQ, a variation of Go-Back-N ARQ optimized for DMA transfers, including its formal analysis. The experimental evaluation included the protocols Stop-and-Wait, Go-Back-N, and DMA ARQ for typical memory traffic scenarios found in real-time systems. The results showed that the simple protocol Stop-and-Wait ARQ suffices for general memory traffic but not for DMA traffic. For DMA transfers, which are bursty in nature, an optimized protocol (DMA ARQ) pays off providing latency bounds 6.6x lower.

5. A Low-overhead Fault-tolerant NoC

The discussion on resilient NoCs for real-time systems has so far focused on ensuring the proper operation of the system under errors. That approach yields a highly reliable NoC that is capable of operating under timing constraints up to high error rates. However, a more lightweight approach that presents smaller overhead but reduced resiliency in comparison to that approach is very attractive if it still is able to meet the requirements of mixed-critical real-time embedded systems. Errors are seldom and the overhead should be minimized following a "good enough" strategy.

Regulated by safety standards [72, 35, 69], such systems must meet strict *real-time*, *resilience* and *integrity* requirements. In that context, threats to the intended functionality of the system must be detected and handled appropriately to meet the specified requirements. In case of errors, threats *must* be detected and contained to ensure integrity; a recovery *might* be performed if possible and if resilience is required for reaching high reliability levels; and all *must* be done in a timely and predictable manner under real-time constraints.

Fast, hardware-based recovery is not always necessary in cross-layer and hierarchical fault-tolerance approaches, where ensuring the system's *integrity* is paramount. In fact, lossless recovery in hardware requires additional circuitry that can introduce substantial power consumption and latencies – e.g., retransmission buffers in ARQ (cf. Chapter 4). Recovery can be performed much more efficiently in higher levels of abstraction, as seen in cross-layer approaches with *replicated execution* [127, 10]. Such techniques exploit the abundant hardware available in multi- and many-core platforms to increase reliability and provide error recovery capabilities in software. Error detection is performed with hardware support, since software-only error detection is ineffective and inefficient. The decision to recover and the error recovery itself are delegated to software.

Nevertheless, the hardware behavior must be predictable since it is a real-time system, and it must detect errors fast enough to allow the system to isolate them and prevent their propagation. That ensures that the recovery, if and as desired, can be carried out in the proper granularity and ensures the integrity of the rest of the system. That reveals two requirements to the hardware operating under soft errors: *integrity* and *real-time* (predictability).

This chapter presents the Advanced Integrity Q-service (AIQ), an end-to-end mechanism to provide integrity and real-time guarantees of NoC transactions under errors. The mechanism is inspired by the idea of keeping track of transactions in distributed systems and Hardware Transactional Memory (HTM). Upon error detection, error handling and recovery are delegated to software, which may react according to an arbitrary strategy in a cross-layer approach. AIQ is proposed and evaluated in a many-core research platform [103] considering aspects such as performance and implementation costs. Although the idea of keeping track of transactions in distributed systems is not novel, to the best of my knowledge, its application in hardware in the context of predictable real-time systems has not been explored and evaluated.

The remaining of the chapter is organized as follows. A review of relevant related work is given in Section 5.1. The AIQ approach is introduced in Section 5.2. Section 5.3 presents the formal analysis of the mechanism. The evaluation is presented in Section 5.4. Finally, Section 5.5 summarizes the chapter.

5.1. Related work

The idea of managing transactions has been widely applied in different fields of computer science and engineering, such as databases, memories and distributed systems. The main objective of those techniques is to ensure the correct execution of memory or database transactions in a concurrent system while increasing performance and parallelism.

In database-management systems [55], transactions are employed to provide the properties of atomicity, consistency, isolation and durability. In that context, the concept is used to increase the level of concurrency, and, thus, performance, in processing numerous simultaneous transactions accessing a single, large database. Thus, transactions might be executed with

speculative data accesses and its changes to the database/system are only committed after a validation phase [85]. Speculative means that read and write operations of different transactions are performed concurrently and are not synchronized, which creates a race condition in the event that two transactions access the same data. If an illegal interleaving of reads and writes to the database occurred due to the race condition, the respective transaction is rolled-back and restarted without changing the state of the system.

Inspired by transactions in database-management systems and existing work on HTM [65], Hammond et al. proposed the Transactional Memory Coherency and Consistency (TCC) as an alternative to traditional memory consistency and coherency models [58]. TCC aimed at simplifying parallel software programming and increasing the concurrent performance of shared memory multiprocessors. Unlike traditional consistency models, accesses to critical sections of the code – i.e., lock-based access to shared memory – must not be explicitly specified, but are carried out by transactions. Transactions are atomic from the point of view of consistency. Unlike traditional coherency approaches, data status synchronization can be performed only at the end of a transaction instead of every memory access[1]. Further extensions to the model include the nesting of transactions [102] – e.g., caused by the use of libraries. A good overview of transactional memories is given in [60]. As of 2015, four processors featured hardware transactional memories: IBM Blue Gene/Q, IBM zEnterprise EC12, Intel Core 4th generation, and IBM POWER8 [106]. In the context of mixed-critical real-time systems, HTM has also been researched. In the context of mixed-critical real-time systems, HTM has been explored as hierarchical HTM on distributed embedded system spanning over on-Chip and off-Chip networks [114].

In the context of fault tolerance, HTM was explored as a hardware-assisted mechanism for error detection and recovery in replicated software execution. The hardware-assisted fault tolerance approach (HAFT) employs instruction-level redundancy for error detection and HTM for recovery with compiler-based code instrumentation [86]. Before committing a transaction to memory, error detection is performed by comparing the instruction-level redundant execution. In case of error, recovery is carried out by rolling-back the transaction with HTM. Similarly, FaulTM also employs instruction-level redundancy and HTM but proposes hardware ex-

[1]The actual operation depends on the actual coherency scheme.

tensions, instead of a software-only approach [159]. Those approaches are types of replicated execution [127, 10]. However error detection requires *all* tasks/threads in the system to be protected and to execute redundantly. A single unprotected task/thread may cause system failure and violate integrity.

In this chapter, the concepts of transactions and ARQ are employed in NoCs to monitor, detect random hardware faults during runtime and ensure system integrity while not jeopardizing system performance. The difference with previous work lies in that all communication in the NoC is covered instead of only replicated tasks, does not require replicated execution and does not depend on the components of a tile – i.e., whether the tile has a processor with caches, only an IP or an external-memory controller. The hardware ensures that any error in a transaction is detected and delegates to software the task of handling the error and choosing the best strategy to do so. The strategy may involve rolling-back, restarting or killing a task/thread or failing, when no other strategy can be safely applied – e.g., as seen in [127, 10]. Upon failure of a task, the effects of the error are isolated in order to ensure the integrity of the rest of the system. Upon failure of the system, the failure should be signaled before any erroneous output is performed.

5.2. The AIQ approach

5.2.1. Overview

The AIQ approach is a NoC service inspired by memory transactions and ARQ-based protocols. AIQ works as a service that keeps tracks of transactions across the NoC with respect to integrity and timing. Figure 5.1 gives an overview of how the approach is integrated into the NoC. The AIQ service operates in the transport layer in the NoC and is located between the interfaces of the NI to the tile internals and the lower layers of NoC. AIQ detects soft and hard hardware errors and reports it to the software layer. The error report signaling path is depicted with the dashed arrows. The AIQ approach delegates to software the task of deciding and recovering from an error in favor of less overhead in hardware. Since recovery is not required, power-hungry retransmission buffers are avoided.

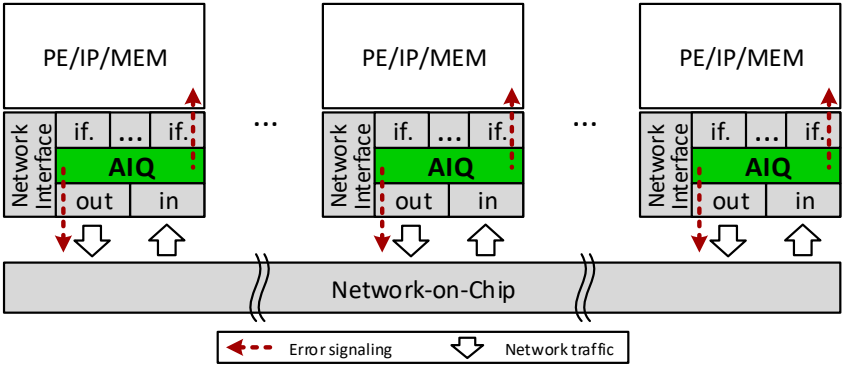


Figure 5.1.: Overview of the AIQ as a NoC service.

5.2.2. Transactions in the NoC

The mechanism keeps track of transactions in the NoC in principle by acknowledging successful transactions. However, given the stringent power and area constraints of small-scale on-Chip networks, a closer look at how transactions take place over the NoC is required.

A regular, unprotected transaction is usually initiated by a *master* who sends a request to a *slave*, which will receive the request and respond after it has been completed. For example, tile $t1$ synchronizes with $t2$ by polling the value of a shared variable in the local memory of $t2$. Figure 5.2a illustrates such a read transaction, where ① $t1$ sends a read request packet and waits for the response. ② $t2$'s NI receives the request packet and forward the request to the tile. After some time, depending on the resource contention in the tile, ③ the response is packed and sent. The response packet is then received by $t1$ ④, which resumes its execution. Notice that the lower layers of the NoC protocol stack are abstracted away for the sake of clarity. Notice also that the same pattern occurs in the case of a write operation or in case of message passing, which may include or not a response (blocking or non-blocking). Whether a response is required depends on the memory and communication models.

The *master* might present overlapping memory transactions depending on the memory consistency model, where support for non-blocking reads and writes exist. The *master* might also support DMA transfers, which

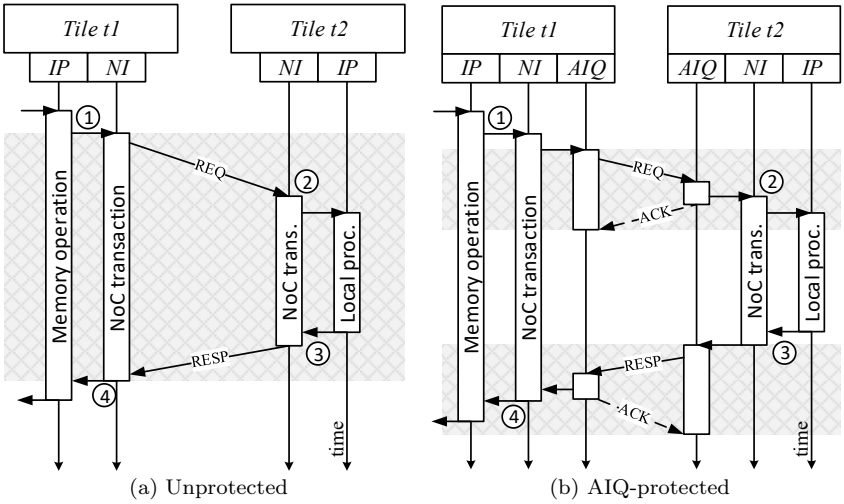


Figure 5.2.: Example of a transaction over the NoC, consisting of a request (REQ) and a response (RESP).

will overlap with regular memory transactions when appropriately used.

A *slave* will potentially receive multiple concurrent requests independently of the memory model since more than one *master* can send a request concurrently. The *slave* can then support single or overlapping requests. In the former case, one request is received and processed at a time, subsequent requests are not accepted by the slave until the completion of the current one. In the latter case, more than one request can be processed at a time. Potentially, in both cases, arriving requests will queue up in the NoC causing backpressure and head-of-line blocking, impacting the arrival of subsequent requests and responses. Resource sharing management is a technique that allows the predictable management of shared resources, as might be required in the *slave*. However, that will not be further discussed here, and the interested reader is referred to [84] for more details. For the sake of simplicity, the discussion in this work assumes single transaction support in *slave*s, unless stated otherwise. That is assumed without loss of generality since the support for multiple concurrent transactions, from AIQ's perspective, would simply generate bursts of packets.

5.2.3. Error model

The transactions are composed of packets, which can suffer a series of different impacts due to random hardware faults in the NoC. A comprehensive description of the impact of soft errors featuring the results of an FMEA-based analysis is given in Chapter 2. AIQ's functional error model was derived based on those results.

On an end-to-end communication stream, as seen by the NIs, the impact of random hardware faults can be summarized as follows (packet and data are used interchangeably):

1. *Data corruption*
   - *Correct delivery of corrupt data*: packet is delivered corrupted to the correct destination;
   - *Incorrect delivery of data*: correct packet is delivered to the wrong destination;
2. *Data loss*: packet is not delivered;
3. *Delayed data delivery*: packet is delivered with abnormally longer latency.

The error-free case and the case where the error has no noticeable impact are intentionally left out. Notice that errors 2 and 3 also capture the case where the NoC becomes partially blocked/unavailable due to a soft error with static effects (cf. Chapter 2).

Hard errors have the same impacts on an end-to-end basis as the ones listed above but with a different occurrence pattern. While soft errors are transient or intermittent in nature and affect traffic randomly according with a probabilistic distribution, hard errors are permanent and, upon occurrence, affect the traffic continuously. Thus, the difference between soft and hard errors is captured in the error model by the frequency in which a certain error effect occurs. The difference between hard errors and soft errors with static effects is that the latter disappear when the NoC is reset.

5.2.4. Protocol

AIQ relies on two well known error detection mechanisms in computer networks: packet integrity check and packet delivery confirmation. The

former is based on EDCs and ECCs and can be realized in hardware, or in some cases delegated to software. The latter is realized in hardware with watchdogs and acknowledgment messages, as seen in ARQ-based protocols.

AIQ aims at decoupling the data transport over the NoC from the processing in the tiles, as depicted in the example of Figure 5.2b. The reason for it is that being predictable and providing timing bounds that include the processing in the tiles creates a complex circular dependency. The traffic in the network depends on the processing in the tiles and vice versa. The AIQ approach should be applicable *without previous knowledge* of the tile internals. Thus, AIQ tracks requests and responses independently in the NoC instead of keeping track of the entire transaction. That is vital for achieving low error detection latencies and effectively limiting the error impact. Moreover, that enables the formal performance analysis and guarantees of the approach.

The objectives of AIQ are:

- Detect all relevant soft and hard random hardware faults in the NoC;
- Operate independently of:
  - The tiles' contents and operation;
  - The NoC topology;
- Report detected errors with very low latencies;
- Minimize the NoC's performance overhead.

AIQ keeps locally a tracking table common for both requests and responses with n entries, as illustrated in Table 5.1. Every request (resp. response) transmitted by the *master* (resp. *slave*) requires an entry in the table. The entry is kept until the sender confirms the successful transmission of the request (resp. response) or until an error affecting the request (resp. response) is detected. Each entry has a sequence number that identifies the request/response; a flag indicating the entry's state; the request/response type – e.g., read response; a timer; the source and destination of the request[2]; and the address of the request/response. Some fields are used for tracking and others enable the diagnosis after error detection.

The AIQ protocol is divided in two parts, as illustrated in the example in Section 5.2.2: a *master* instance for sending requests and a *slave* instance

[2]The source identifies the interface generating the request/response (cf. Figure 5.1). The destination can be deducted from the accessed address by accessing the NI's address translation table. Alternatively, it can be stored as VC+route.

Table 5.1.: AIQ request/response tracking table

Tracking				Diagnosis	
Entry	**State**	**Type**	**Timer**	**Src/Dst**[2]	**Address**
1	Used	Req1	...	0	0xaffd0000
...	...	...	...	...	...
n	Free	...	0	0	0x00000000

for sending responses. A summary of the different scenarios is given in Table 5.2. They are described individually in the sequel.

Table 5.2.: AIQ error notification scenarios

	REQ		**RESP**	
	loss	**corrupt**	**loss**	**corrupt**
Sender (*master*)	✓	✓	✓	✓
Receiver (*slave*)		✓	OPT	OPT

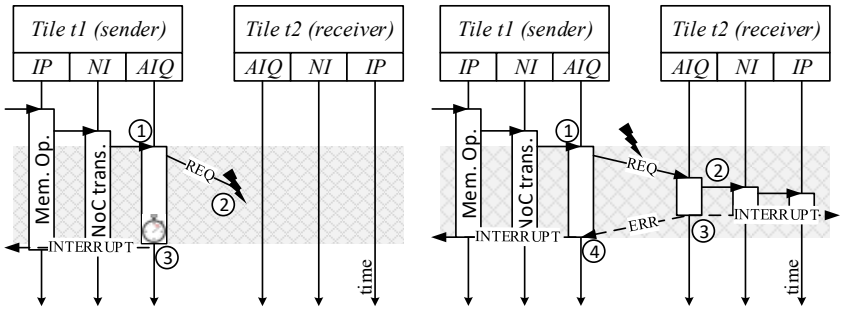
5.2.4.1. Requests

AIQ must account for two cases when tracking requests: loss and corruption.

Loss is monitored with a handshaking mechanism, as seen in ARQ protocols. When the request is sent, the timer associated with the request is triggered. When the request is correctly received, the *slave*'s AIQ sends an ACK packet back to the sender. If the ACK is correctly received, the respective timer is stopped and the request is marked as not pending, releasing the respective entry. The scenario is illustrated in Figure 5.2b. If the ACK is not received, a timeout will trigger the error detection activities. That occurs if the request itself was lost or if the ACK was lost. In the former case, the *slave* is unaware of the failed request and remains unaffected by the error. That is illustrated in Figure 5.3a. In the latter case, the *slave* received the request and will process it correctly whereas the *master* cannot be certain that the request was received[3]. Thus, in both cases, only the *master* is notified with a hardware interrupt in order to take appropriate action.

Upon loss detection, four actions are carried out at the *master*:

[3]The *master* must be able to handle the "orphaned" responses.



(a) Case 1: loss (b) Case 2: corruption

Figure 5.3.: Illustration of error affecting requests (REQ).

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- the respective request table entry is released;
- the interface that issued the affected request is notified to abort the transaction in order to allow the further operation of the system;
- and a hardware interrupt is triggered so that appropriate action can be taken in software.

For example, fault containment can be performed by the Real-Time Operating System (RTOS) and error recovery can be performed, e.g., by a replica manager, as seen later in Chapter 6. Unaffected tasks may continue executing normally.

Corruption is verified with an integrity check using EDC and possibly ECC, both at the *slave* and at the *master*. The integrity check is mandatory for control fields and optional for data[4]. Requests with corrupt control fields are immediately dropped to keep the integrity of the node – e.g., prevents unintended access and modification (corruption) of memory contents. As illustrated in Figure 5.3b, upon integrity of control fields ② the request is forwarded to the *slave* tile. The integrity check of the data (when enabled) is performed on-the-fly as the data is forwarded to the tile. That improves performance and reduces hardware overhead by avoiding the use of large buffers. The result of the integrity check is available when the last data word of the request traverses the AIQ. Thus, possibly part of

the data might have already reached the tile (e.g., memory) by the time the corruption is detected and signaled. Nonetheless, the signaling ③ will occur before the last word of the request leaves the *slave*'s NI. The error is also signaled back to the *master* ④ with a NACK packet.

Upon corruption detection, the following actions are carried out at the *slave*:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- the interface receiving the affected request is notified to abort the transaction in order to allow the further operation of the system;
- a hardware interrupt is triggered locally so that appropriate action can be taken;
- a NACK is sent to the *master* in order to trigger the error detection actions.

Upon the receipt of a NACK, the following actions are carried out at the *master*:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- the respective request table entry is released;
- the interface receiving the affected request is notified to abort the transaction in order to allow the further operation of the system;
- a hardware interrupt is triggered locally so that appropriate action can be taken in software.

In case the NACK is not successfully received, e.g. due to the failure of the NoC, the case will be handled as a request loss.

5.2.4.2. Responses

Similarly to requests, AIQ needs to account for two cases when tracking responses: loss and corruption. In both cases, error detection occurs with the same approach and mechanisms as for requests. The difference lies in

[4]It is optional for data for cases where the written data will be checked for integrity already in software. The data integrity check can be configured during runtime in a memory range granularity, extending the configuration capabilities of the IDAMC platform. The address is considered a control field and its integrity check is mandatory.

the error reporting.

The *loss* of a response must be reported back to the *master*. It can be optionally[5] reported locally to the *slave*. Upon the loss detection, the following actions are carried out at the *slave*:

- a NACK is sent to the *master* in order to trigger the error detection actions; the NACK will be transmitted following Stop-and-Wait ARQ.

Optionally, the following actions can be carried out at the *slave* to trigger a reaction locally:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- a hardware interrupt is triggered locally so that appropriate action can be taken.

Upon the receipt of a NACK, the following actions are carried out at the *master*:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- the interface receiving the affected request is notified to abort the transaction in order to allow the further operation of the system;
- a hardware interrupt is triggered locally so that appropriate action can be taken in software;
- the NACK is acknowledged (Stop-and-Wait ARQ).

The *corruption* of a response is detected and reported locally to the *master*. Upon corruption detection, the following actions are carried out at the *master*:

- the information necessary for error diagnosis is stored in dedicated registers in the AIQ;
- the interface receiving the affected request is notified to abort the transaction in order to allow the further operation of the system;
- a hardware interrupt is triggered locally so that appropriate action can be taken.

[5]The lost response can be used to detect the failure of the NoC due to hard errors or static effects, in which case the *master* will most likely fail to receive the NACK. The tile contents are, however, unaffected by the loss and requires no further action.

5.2.5. Discussion

Due to the size and resource restrictions in a NoC in comparison with an off-Chip network, two points were identified during implementation that can impact the correct operation and the performance of the approach. The identified points concern the limitations of the interfaces of the NI in handling received requests and the limitations of the input buffers in the physical and data-link layers of the NI.

When handling received requests, the NI handles each request sequentially instead of handling them in parallel or buffering the requests, which would be too costly and unnecessary in most cases. Thus, subsequent requests and packets can potentially be blocked due the backpressure. Figure 5.4 shows a block diagram of the NI including the AIQ mechanism (corruption detection with CRC is shown separately). The backpressure is illustrated with the red arrow ①. A request is received by the *Mst_if* interface and blocks the following packets as indicated by the arrow. That interface will only accept a next request after the current one has been served. That can make the timing of the approach dependent on the workload and on the internal details of the tile. The key impact on AIQ is that the blocking introduces additional delay to requests/responses, which are expected to be acknowledged as soon as they arrive at the NI. Thus the latency becomes dependent not only on the NoC topology and interfering traffic, but also on the internal performance of the tile.

The first point is addressed in this work by bounding the maximum time that an interface in the NI can take to process a request. The bound must be realized by the NI and tile designs – e.g., the NI might abort the request if it is not completed by the tile within the specified bound; alternatively the tile can be designed to accept multiple concurrent requests.

The second identified architectural limitation is related with the NI's input buffer (*buffer phit2flit*), which reassembles flits from phits and buffers them until they can be forwarded to the upper layers of the protocol stack. The flits of different VCs are stored in different queues. It can happen that a control message of AIQ (ACK or NACK) experiences head-of-line blocking depending on the arbitration policy and depending on the type of packets queued in front of it. That is illustrated by arrow ② in Figure 5.4. The situation can escalate to a deadlock, which will be detected as a NoC failure, since no ACKs are being received by AIQ. The deadlock occurs when the *Mst_if* is ready to respond a request but no entries are available

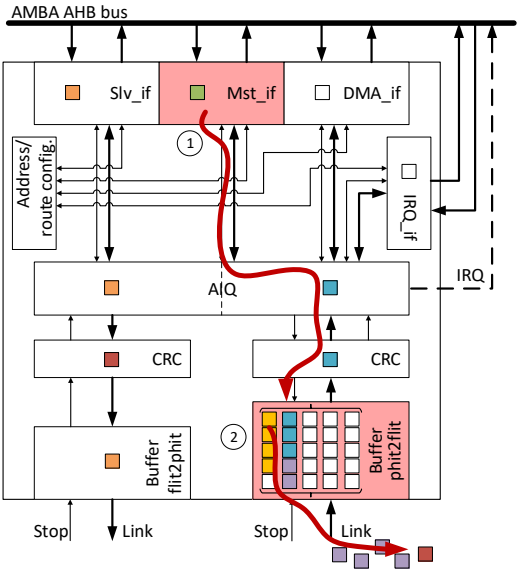


Figure 5.4.: Block diagram of NI: identified limitations.

in AIQ's tracking table (cf. Table 5.1). The ACK that releases a table entry is then blocked in the *buffer phit2flit* due to backpressure from *Mst_if*. Thus, *Mst_if* can only progress when an ACK is received but the ACK is blocked by *Mst_if*.

The deadlock can be ruled out by using separate VCs for control packets and for requests/responses. Alternatively, it can be ruled out by ensuring that the number of concurrently received requests in a tile is limited and do not block the control packets. That can be achieved by using a resource manager, such as [84], which manages the concurrent access of applications to shared resources in the platform. This work adopts the former solution.

5.3. Formal analysis of AIQ

As seen in Chapter 4, transport protocols, such as Go-Back-N ARQ and the proposed AIQ, introduce additional flow control to the communication. That comes from packets that are retained due to handshaking or due to

retransmissions with timeouts (in ARQ). It creates a circular dependency where the performance of the transport layer depends on the network latency, which in turn depends on the traffic injected by the transport layer. In this section, AIQ is modeled using CPA [62] based on the transport layer analysis introduced in Chapter 4.

The formal timing analysis of AIQ is presented in three parts. First, the modeling in CPA is introduced. Then, the protocol behavior in the error-free scenario in analyzed. Finally, the error case is addressed with an analysis of the worst-case latency in error reporting.

5.3.1. Modeling in CPA

The modeling in CPA, following the terminology introduced in Section 4.2.1, is illustrated in Figure 5.5. An interface in the sender's NI produces packets in a traffic stream according with a given packet-based event model δ_{tx}. The packets are handled at the transport layer by AIQ, which is modeled as a resource. The analysis assumes that packets transmitted from different interfaces within a NI are arbitrated according with Strict Priority Non-Preemptive (SPNP) and also assumes that ACKs and NACKs have the highest priority. Each interface producing packets is modeled as a task τ mapped to that resource – the interface under analysis is depicted as τ_{aiq} whereas a lower priority one and a higher priority one are depicted as τ_{lp} and τ_{hp}, respectively. Packets generated by AIQ are captured by a dedicated task – e.g., ACKs transmitted by the sender NI are captured by τ_{ack}. AIQ then injects traffic into the lower layers of the NoC according to the output event model δ_{aiq}. Notice that only one interface (or AIQ itself) can transmit a packet at a time and therefore only one output traffic stream is depicted for the sender even though several traffic streams can co-exist in the NoC and possibly originate in the same sender.

A protocol with handshaking, such as AIQ, is a bidirectional communication stream [130]. As illustrated in Figure 5.5, the communication is mapped in the NoC as two unidirectional streams: one for data and one for acknowledgements in the feedback path. As in Chapter 4, the underlying NoC analysis is arbitrary. This analysis assumes, without loss of generality, [125] as the underlying NoC analysis, which models the NoC in CPA as follows: each output port of a router is mapped as a resource, and traffic streams are chains of tasks mapped to resources. Resource arbitration depends on the router arbitration. The output of the underlying NoC

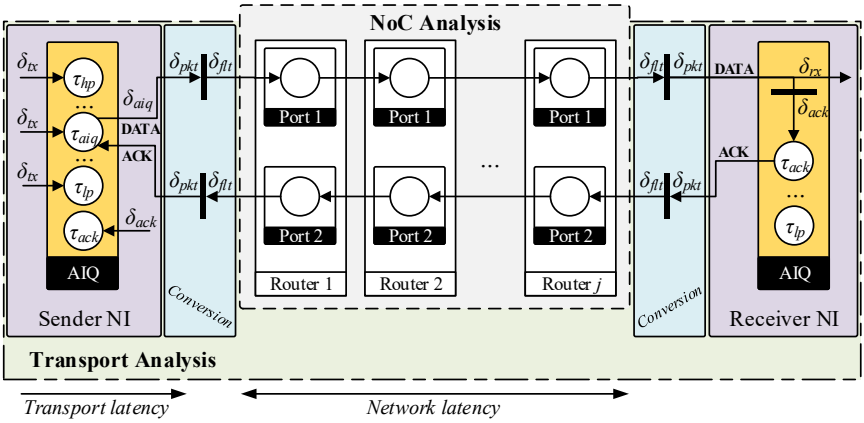


Figure 5.5.: Modeling of AIQ as a transport layer protocol and the underlying NoC in CPA.

analysis used by the transport analysis is the worst-case latency L_i^+ of a packet transmitted in a traffic stream i. The interested reader can refer to [125, 130] for further details.

The analysis supports both packet-switched and wormhole-switched NoCs. Packet-switched NoCs are supported by default. For wormhole-switched ones, however, a conversion between event models is necessary, as seen in Chapter 4. That is depicted by the light blue elements in Figure 5.5. The conversion between packet-based and flit-based event models can be performed with the Equations 4.3 and 4.4 of Section 4.2.2.

5.3.2. Formal analysis: the error-free case

At first sight, the timing behavior of AIQ seems similar to Go-Back-N ARQ, seen in Chapter 4. However, AIQ differs from ARQ in that one instance in a NI is shared among traffic streams whereas, in the latter, each traffic stream has its own protocol instance. That makes a big difference. The latter simplifies the analysis by exploiting the fact that all worst-case processing delays and RTTs are the same for the same traffic stream. In the former, worst-case processing delays and RTTs of interfering packets are potentially different, resulting in the more complex problem of multi-server

queues [155, 94]. The analysis of multi-server queues is a hard problem, with a worst-case that is difficult to tightly bound, and it is thus usually handled with Queueing Theory [79, 155, 94].

The worst-case analysis of AIQ can be simplified in four ways: ① by assuming that there is only one entry in the tracking table (cf. Table 5.1); ② by assuming that there are more entries than packets being transmitted (unlimited number of entries), but where the maximum number of entries required can be bound; ③ by assuming that all traffic streams have the same delays (RTTs and processing delays, which is related to the packet sizes); or ④ by assuming that there is no local interference from other traffic streams. ① and ③ leads to unrealistically pessimistic results. ④ assumes a scenario that is simply unrealistic in practice. Thus, the adopted strategy is ②, which might not be a match in all cases, but allows one to find out the number of entries required to achieve the bounded performance. Similar analysis approaches are seen in the literature [33, 125].

The analysis also assumes that packets transmitted from different interfaces within a NI are arbitrated according with SPNP (cf. Figure 5.1). ACKs have the highest priority.

To obtain the worst-case end-to-end latency of a packet protected by AIQ it is necessary to derive the interference of other traffic in the NI and the contribution of the AIQ protocol to the latency. That is captured by the WCRT of AIQ $R^+_{aiq,i}$, which is the largest period of time in which a packet is retained by the protocol. The analysis relies on the busy window approach [149]. The first step is to derive the Worst-Case Multiple Packet Queuing Delay.

The Worst-Case Multiple Packet Queuing Delay $Q^+_{aiq,i}(q)$ is the longest time interval from the arrival of the first packet until the q-th packet receives service. The $Q^+_{aiq,i}(q)$ of stream i is given by:

$$Q^+_{aiq,i}(q) = I^+_{lp,i} + I^+_{hp,i}(Q^+_{aiq,i}(q)) + I^+_{acks,i}(Q^+_{aiq,i}(q)) + (q-1)\cdot O^+_{aiq,i} \quad (5.1)$$

where

$$I^+_{lp,i} = \max_{j \in lp(i)} \{O^+_{aiq,j}\} \quad (5.2)$$

$$I^+_{hp,i}(\Delta t) = \sum_{j \in hp(i)} \eta^+_{tx,i}(\Delta t) \cdot O^+_{aiq,j} \quad (5.3)$$

$$I^+_{acks,i}(\Delta t) = \sum_{j \in acks} \eta^+_{tx,i}(\Delta t) \cdot O^+_{aiq,ack} \quad (5.4)$$

and where $O^+_{aiq,j}$ is the maximum time that AIQ requires to forward a packet of stream j; $O^+_{aiq,ack}$ is the maximum time that AIQ requires to create and forward an ACK; $lp(i)$ and $hp(i)$ are the set of all lower and higher priority streams mapped to the same AIQ as stream i, respectively; and $\eta^+_{tx,i}(\Delta t)$ is the maximum event arrival curve (cf. Section 4.2.1). Equation 5.1 results in a fixed-point problem. It can be solved iteratively, starting with a very small, positive ϵ.

Lemma 5.1. *Equation 5.1 gives an upper bound on the Worst-Case Multiple Packet Queuing Delay* $Q^+_{aiq,i}(q)$.

Proof. The proof is by induction. When $q = 1$, stream i's packet can be blocked by one non-preemtable, lower priority packet that just started transmitting, assumed to be the largest one causing the longest delay; while queued, the packet can also be blocked by arriving higher priority packets; additionally, the packet can also be blocked by ACKs, which are sent with highest priority, generated due to receiving packets. That is captured in Equation 5.1 by the terms $I^+_{lp,i}$, $I^+_{hp,i}(Q^+_{aiq,i}(q))$, and $I^+_{acks,i}(Q^+_{aiq,i}(q))$, respectively.

In a subsequent $q+1$-th activation, the packet has to additionally wait for its own previous q packets to be forwarded. That takes $O^+_{aiq,i}$ per packet, potentially increasing the interference caused by higher-priority packets and acknowledgements already captured by $I^+_{hp,i}(Q^+_{aiq,i}(q+1))$ and $I^+_{acks,i}(Q^+_{aiq,i}(q+1))$, respectively. That results in Equation 5.1. □

The Best-Case Multiple Packet Queueing Delay $Q^-_{aiq,i}(q)$ is the shortest time interval from the arrival of the first packet until the q-th packet receives service. It is given by:

$$Q^-_{aiq,i}(q) = (q-1) \cdot O^-_{aiq,i} \tag{5.5}$$

where $O^-_{aiq,i}$ is the minimum time that AIQ requires to forward a packet of stream i.

Lemma 5.2. *Equation 5.5 gives a lower bound on the Best-Case Multiple Packet Queueing Delay* $Q^-_{aiq,i}(q)$.

Proof. The proof is by induction. When $q = 1$, stream i's packet can be forwarded as soon as it arrives. In a subsequent $q+1$-th activation, the

packet must wait for the previous q packets to be forwarded. That results in Equation 5.5. □

The Worst-Case Multiple Packet Forwarding Time $B^{+}_{aiq,i}(q)$ is the longest time interval from the arrival of the first packet until the forwarding of the q-th packet. It extends $Q^{+}_{aiq,i}(q)$ until the q-th activation completes. The $B^{+}_{aiq,i}(q)$ of stream i is given by:

$$B^{+}_{aiq,i}(q) = Q^{+}_{aiq,i}(q) + O^{+}_{aiq,i} \tag{5.6}$$

Lemma 5.3. *The Worst-Case Multiple Packet Forwarding Time $B^{+}_{aiq,i}(q)$ given by Equation 5.6 is an upper bound.*

Proof. The proof is by direct deduction. Under SPNP, the time to forward q packets corresponds to the time until the q-th packet is about to receive service and the time it takes to forward the q-th packet, which is non-preemtable. That is captured by the first and second terms of Equation 5.6, respectively. Both terms are upper bounds and Equation 5.6 is thus an upper bound. □

The Best-Case Multiple Packet Forwarding Time $B^{-}_{aiq,i}(q)$ can be similarly derived. It is the shortest time interval from the arrival of the first packet until the q-th packet is forwarded. $B^{-}_{aiq,i}(q)$ is given by:

$$B^{-}_{aiq,i}(q) = Q^{-}_{aiq,i}(q) + O^{-}_{aiq,i} \tag{5.7}$$

Lemma 5.4. *The Best-Case Multiple Packet Forwarding Time $B^{-}_{aiq,i}(q)$ given by Equation 5.7 is a lower bound.*

Proof. The proof is omitted. It is similar to Lemma 5.3, but using lower bounds instead. □

The busy period $w_{aiq,i}$ is the longest time interval in which packets of stream i arrive at AIQ before the previous packet has been transmitted. That is, it is a half-open interval starting with the first activation and ending when activation q completes before the arrival of the $q{+}1$ activation. The busy period $w_{aiq,i}$ is given by:

$$w_{aiq,i} = \max_{q \geq 1, q \in \mathbb{N}} \left\{ B^{+}_{aiq,i}(q) \,|\, Q^{+}_{aiq,i}(q{+}1) \geq \delta^{-}_{tx,i}(q{+}1) \right\} \tag{5.8}$$

Lemma 5.5. *The busy window is upper bounded by Equation 5.8.*

Proof. The proof is by contradiction. Suppose there is a busy window $\breve{w}_{aiq,i}$ longer than $w_{aiq,i}$. In that case, $\breve{w}_{aiq,i}$ must contain at least one activation more than $w_{aiq,i}$, i.e., $\breve{q} \geq q+1$. From Equation 5.8, $Q^{+}_{aiq,i}(\breve{q}) < \delta^{-}_{tx,i}(\breve{q})$, i.e., $\breve{q}$ is not delayed by the previous activation. Since that violates the definition of a busy window, the hypothesis must be rejected. □

The Worst-Case Response Time $R^{+}_{aiq,i}$ is the longest time interval that any packet of a stream i is delayed by AIQ before being forwarded to the network. It is bounded by:

$$R^{+}_{aiq,i} = \max_{1 \leq q \leq \eta^{+}_{tx,i}(w_{aiq,i})} \{B^{+}_{aiq,i}(q) - \delta^{-}_{tx,i}(q)\} \tag{5.9}$$

Theorem 5.1. *$R^{+}_{aiq,i}$ (Equation 5.9) provides an upper bound on the response time of an arbitrary packet in the traffic stream i transmitted under AIQ.*

Proof. Proof omitted due to similarity with proof of Theorem 4.1. □

The event model capturing the traffic injection of stream i in the network by AIQ can now be derived. The output event model $\delta^{-}_{aiq,tx,i}$ propagated by a NI with AIQ is obtained as follows:

$$\delta^{-}_{aiq,tx,i}(q) = \max\left\{\delta^{-}_{tx,i}(q) - R^{+}_{aiq} + R^{-}_{aiq}, B^{-}_{aiq}(q-1)\right\} \tag{5.10}$$

Theorem 5.2. *The minimum distance function $\delta^{-}_{aiq,tx,i}(q)$ given by Equation 5.10 is a lower bound.*

Proof. Proof omitted due to similarity with proof of Theorem 4.2. □

The time it takes to transfer q packets can now be bounded, where q might range from a single small packet to a long DMA transfer[6]. The *overall latency* $\mathbb{L}^{+}_{aiq,i}(q)$ of transmitting q data packets in a stream i is given by:

$$\mathbb{L}^{+}_{aiq,i}(q) = \delta^{-}_{tx,i}(q) + L^{+}_{i} + R^{+}_{aiq,i} \tag{5.11}$$

where L^{+}_{i} is the worst-case NoC latency of any packet in stream i (cf. Section 4.2.3).

[6]DMA transfers can consist of a single, very long packet or multiple smaller packets depending on the NoC architecture and configuration

Theorem 5.3. *Equation 5.11 gives an upper bound on the overall latency to transmit q data packets under AIQ.*

Proof. Proof omitted due to similarity with proof of Theorem 4.3. □

Finally, the time it takes to perform a NoC transaction consisting of request and response under AIQ can be bounded. The *transaction latency* $\mathbb{L}^{+}_{trans}(q_{req}, q_{resp})$ of a transfer comprising q_{req} request packets and q_{resp} is given by:

$$\mathbb{L}^{+}_{trans}(q_{req}, q_{resp}) = \mathbb{L}^{+}_{aiq,req}(q_{req}) + O^{+}_{proc} + \mathbb{L}^{+}_{aiq,resp}(q_{resp}) \qquad (5.12)$$

where the request and the response consist of q_{req} and q_{resp} packets, respectively, and O^{+}_{proc} is an upper bound of the time it takes for the transaction to be processed and responded (see Section 5.2.5).

Theorem 5.4. *Equation 5.12 gives an upper bound on the overall latency to complete a transaction.*

Proof. The proof is by direct deduction. The latency of a transaction consists of the latency to transmit the request, the time it takes for the receiver to process the request and generate a response, and the latency to transmit the response. That is captured by the first, second and third terms of Equation 5.12. From Theorem 5.3, the first and third terms are upper bounds. The second term (O^{+}_{proc}) is an upper bound by definition. Thus, Equation 5.12 is a valid upper bound. □

5.3.3. Formal analysis: the error case

In this section, AIQ is analyzed with respect to its error detection latency guarantees. In contrast to the ARQ-based protocols, introduced in Chapter 4 and which guarantee packet delivery, AIQ does not provide error recovery and thus does not introduce itself additional latency due to errors. Error recovery might be performed in software and will certainly incur additional processing time, whose worst-case behavior under errors has been analyzed, e.g., by [127]. As summarized in Table 5.2, two cases must be detected by AIQ – loss and corruption. Upon detection, the error must be notified to the local tile or to the remote tile depending on whether the the affected packet was a request or a response.

The worst-case impact of an error on the detection latency is when the error causes a request/response loss, where the detection of a packet loss occurs upon the timeout event of a timer. In contrast with the detection of corruption, which occurs at the arrival of a request/response, the detection of packet loss will always take longer due to the timeout. Such worst-case error impact is similarly seen in ARQ-based protocols (cf. Chapter 4 and Appendix A). In the sequel, the worst-case detection latency is analyzed with respect to transient faults. The impacts of permanent faults and permanent effects is discussed afterwards.

In case of request loss, only the local tile must be notified (cf. Table 5.2). The Worst-Case Error Detection Latency for local reporting $\mathbb{L}^{+err}_{aiq,i}(q)$ is the longest time interval between the transmission of a request on stream i until the notification that its transmission on the NoC failed. It is given by:

$$\mathbb{L}^{+err}_{aiq,i}(q) = \delta^{-}_{tx,i}(q) + R^{+}_{aiq,i} + t_{out,i} + O^{+}_{aiq,int} \tag{5.13}$$

where $t_{out,i}$ is the timeout value for the request packet of stream i and $O^{+}_{aiq,int}$ is the maximum delay from timeout detection until a hardware interrupt is raised. Similar to ARQ protocols, the timeout must be chosen larger than the worst-case RTT, usually including a safety margin – i.e., $t_{out,i} > RTT^{+}_{i}$ (cf. Definition 4.2).

In case of a response, a remote notification with a NACK is required, which extends the notification latency. That is captured by the Worst-Case Error Detection Latency for remote reporting $\mathbb{L}^{+err\,rem}_{aiq,i}(q)$ and is given by:

$$\mathbb{L}^{+err\,rem}_{aiq,i}(q) = \delta^{-}_{tx,i}(q) + R^{+}_{aiq,i} + t_{out,i} + L^{+}_{nack} + O^{+}_{aiq,int} \tag{5.14}$$

where $t_{out,i}$ is the timeout value for the response packet of stream i, and L^{+}_{nack} is the worst-case NoC latency of the NACK packet (cf. Section 4.2.3).

It is possible that the NACK is delivered to the *master* only after retransmission attempts, which can occur in *multiple error scenarios* in very high error rates. In that case, $k \cdot t_{out,NACK}$ can be appended to Equation 5.14 to account for the k additional retransmissions with timeout $t_{out,NACK}$.

In case of permanent faults or transient faults with static effects causing the failure of the NoC, it is possible that the NACK is not delivered at all. AIQ is also able to detect NoC failures by monitoring the frequency of error occurrences and by detecting the failure of a remote notification. In case of network failure, a dedicated error single-wire signal, shared among

all nodes, can be employed to notify the failure to the otherwise unreachable system controller. The controller can then reset the NoC, which will cause the unavailability of the NoC for some time, called Mean Down Time (MDT), whose length depends on the hardware/software implementation. Moreover, the reset of the NoC must be carried out in such a way that the remaining transactions are allowed to finish, so that only the tasks whose transactions failed due to an error will trigger recovery in software. Otherwise, a the reset could induce the failure of all pending transactions and lead therewith to an undesirable scenario.

5.4. Experimental evaluation

AIQ has been evaluated with respect to performance, implementation overhead, and the achieved reliability and availability. The main objective of the experiments is to evaluate the impact of AIQ on the regular performance of the MPSoC. On the impact of errors on NoCs, the interested reader can refer to the investigations of Chapter 2.

The performance was evaluated with the many-core platform IDAMC [103]. Benchmark applications as well as an avionics use case were executed on two versions of the platform: a baseline version; and a version with AIQ. Moreover, two different mapping configurations are used to stimulate an extreme application scenario and one expected application scenario. In the first scenario, the applications are executed remotely inducing the direct impact of the NoC latencies on the application performance – i.e., the application code and data are mapped to memory in remote tiles. In the second scenario, the application nodes execute locally, emulating a Logical Execution Time execution model with inter-core communication for synchronization and DMA for data transfers – i.e., code and data mapped to local memory, and shared memory communication and code download from memory in remote tiles. The two scenarios provide a valuable contrast between AIQ's impact on the NoC traffic and the impact of the NoC performance on the application's execution. Finally, the hardware implementation overhead of AIQ is evaluated, followed by a reliability assessment and discussion. The results presented in this section regard a VHDL design of the IDAMC platform and AIQ simulated in RTL with QuestaSim [101], synthesized for FPGA with Xilinx Vivado [157] and synthesized for CMOS ASICs with Synopsys Design Compiler [145].

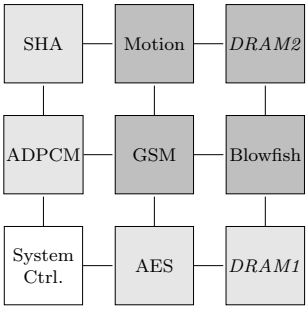


Figure 5.6.: Mapping of the CHSTONE benchmark applications to the 3x3 NoC.

5.4.1. Performance evaluation: benchmark applications

The performance impact of AIQ was first evaluated with CHSTONE benchmarks [59]. The benchmark applications were mapped to the IDAMC platform as depicted in Figure 5.6. The applications' code and data were mapped to remote a remote memory, according with the first of the aforementioned mapping configurations. In the setup, application tiles generate traffic due to cache misses and evictions and due to uncached data accesses. The applications were divided in two groups: one group (light gray) accesses the memory in tile *DRAM1* and the other group (dark gray) accesses the memory in tile *DRAM2*. The NoC is configured with XY routing for requests, YX routing for responses, and a separate VC for each application. The interested reader can refer to the overview in Chapter 1 for more details on the IDAMC and on the NoC.

The results of the RTL simulations can be seen in Figure 5.7, which plots the latencies of NoC transactions of the different applications as boxplots. Furthermore, the plot compares latencies in an unprotected NoC (base) with latencies in a NoC protected with AIQ. In a boxplot, the whiskers represent the maximum and minimum, the box represents the second and third quartiles, the horizontal line indicates the median value, and the marker indicates the mean value. First, the minimum latency increased 9 cycles in all applications. That is due to the increase in the pipeline length in the NI. As seen in Figure 5.4, by introducing AIQ and the CRC checker, the pipeline was extended by four stages for a request and for a response. On average, the latencies increased 16.1% across all applications. That is

caused by the increased pipeline length as well as the additional feedback traffic consisting of ACKs.

This experimental setup intentionally induces a high amount of traffic whose performance strongly impacts the execution time of the applications. Thus, execution time increase varied from 10.8% (*ADPCM*) up to 15.5% (*Motion*), depending on the application's memory footprint. On average, the execution times increased 12.6% across all applications.

Detailed results of the *SHA* application are shown in Figure 5.8. The figure plots the trace of latencies of NoC transactions in time, comparing the cases with the unprotected and the protected NoC. In the protected NoC, the latencies are slightly longer and show a higher variance than in the unprotected one due to the additional feedback traffic introduced by AIQ. In fact, the average latency increases in about 14.9% from 94.94 to 109.11 clock cycles (cf. Figure 5.7). Due to high impact of the NoC latencies on application performance artificially induced by the experimental setup, the execution time of *SHA* increased 12.66%. That can be considered as upper bound for the overhead caused by AIQ on cache-enabled executions. As seen next, the impact on performance of executions with local memory is much lower.

5.4.2. Performance evaluation: avionics use case

AIQ was also evaluated with a parallelized avionics application. Due to the confidentiality of industrial developments, the experiments employ an Artificial Demonstration Application (ADA) that mocks the dataflow and workload of a Helicopter Terrain Awareness and Warning System

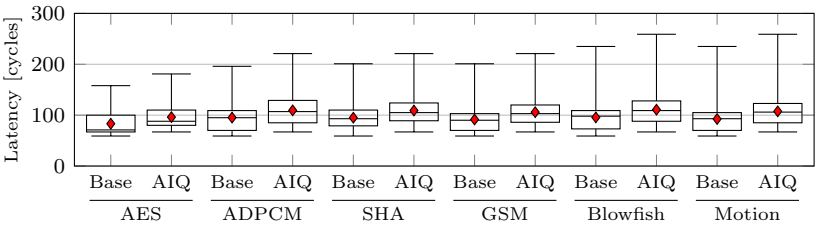
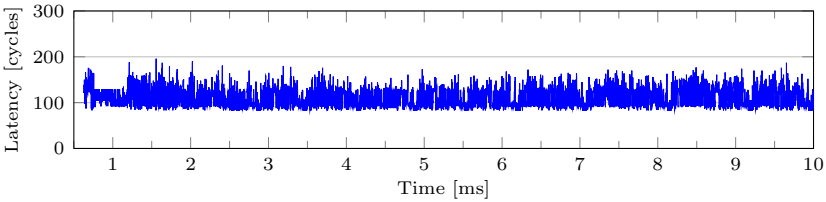
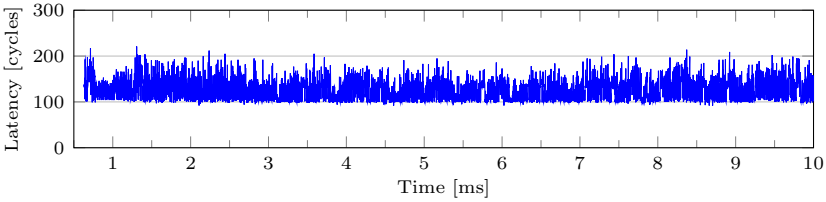


Figure 5.7.: Boxplot of observed latency of NoC transactions of CHSTONE benchmarks in a protected (AIQ) and unprotected NoC (Base).



(a) Unprotected

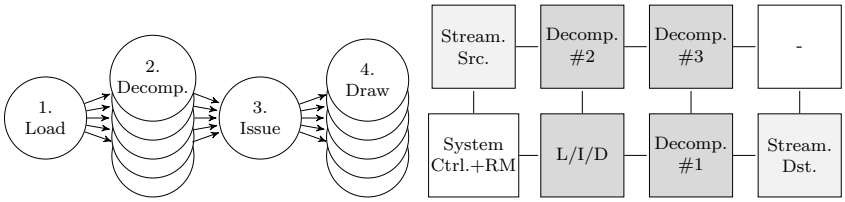


(b) Protected with AIQ

Figure 5.8.: Latency of transactions of the *SHA* application executing on an unprotected NoC and on a NoC protected with AIQ.

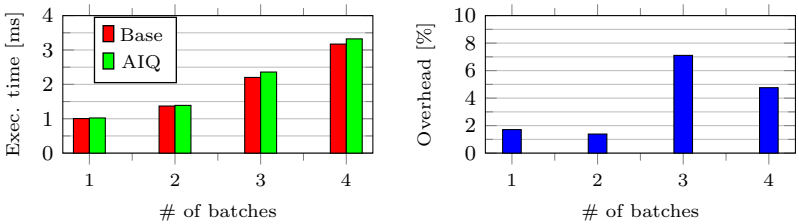
(HTAWS) [44]. The HTAWS consists of multiple threads with DAL-C [35] executing on an SMP with an RTOS. The main application dataflow, depicted in Figure 5.9a, comprises four major pipeline stages. Two of the stages (*Decomp.* and *Draw*) can be parallelized to increase performance by exploiting the available data parallelism. A major frame must be processed from input to output in at most 60ms. In the original single core application, the four stages are executed sequentially, with a period of 60ms. In the parallel version, the stages are executed in a pipeline to increase the overall throughput.

The application is mapped to a 2x4 instance of IDAMC, as depicted in Figure 5.9b. Stages 1, 3 and 4 are mapped to a single tile (*L/I/D*). Three instances of stage 2 are mapped to different tiles (*Decomp. #1*, *#2* and *#3*). Additionally, interference is introduced in the platform by node *Stream. src*, which generates DMA traffic to node *Stream. dst* with 4.8KB DMA transfers. The system controller (*System Ctrl.+RM*) initializes the platform and manages the access to shared network resources by implementing a resource manager [84].



(a) ADA application. (b) Mapping on 2x4 NoC.

Figure 5.9.: ADA application and its mapping to a 2x4 IDAMC, including interfering applications.



(a) Execution time (b) Overhead in time

Figure 5.10.: Response time comparison of ADA with a protected NoC (AIQ) and an unprotected one (Base).

Figure 5.10 reports the execution time of ADA under different setups in RTL simulations of the IDAMC platform. When increasing the workload, specified as number of batches, ADA takes between 1ms and 3.2ms to execute on IDAMC with an unprotected NoC (Base). With a NoC protected with AIQ, the execution takes from 1.4% to 7.1% longer (cf. Figure 5.10b). In contrast to the benchmark applications evaluated in the previous section, these executions with AIQ present much lower overhead. That is due to the more realistic setup expected in real-time multi- and many-core platform environments, where a clearer separation between computation and communication is required to limit interference, to achieve predictability and to avoid prohibitive analytical over-approximations. That trend can be seen in predictable execution models such as superblocks [138, 117].

Figure 5.11 reports the latencies of NoC transactions observed in the

scenario with 4 batches. The plot shows, as boxplots, the observed latencies of NoC transactions initiated by different ADA application nodes with and without AIQ. The average transaction latency increases between 14.8% and 19.7% with AIQ. The minimum latency increases 15.5% due to the additional cycles required by the extended pipeline in the NIs. The observed maximum, however, decreased between 3.9% and 7.8%, i.e., between 4 and 13 clock cycles. That is due to a slightly different network traffic patterns that result from the interaction with the feedback traffic and that induces a scenario where some packets to experience less interference. Nonetheless, one can observe that those scenarios are exceptional as latency increases are the expected case.

5.4.3. Implementation overhead

The implementation overhead of AIQ was evaluated for FPGAs and ASICs. The evaluation features a baseline NI and NIs protected with different configurations of AIQ. For FPGAs, the designs were synthesized in Xilinx Vivado [157] targeting a Virtex-7 FPGA (xc7v2000tflg1925-1) and a frequency of 80MHz. For CMOS ASICs, the designs were synthesized in Synopsys Design Compiler [145] with UMC 65nm cell libraries (high and low threshold voltage, worst-case corner 0.9V 125C) targeting a frequency of 125MHz; and with TSMC 28nm cell libraries (high and standard threshold voltage, worst-case corner 0.72V 125C) targeting a frequency of 1GHz. The evaluation involved a NI configured with four interfaces:

- *DMA if.* for DMA transfers;
- *Mst if.* for remote NoC transactions;

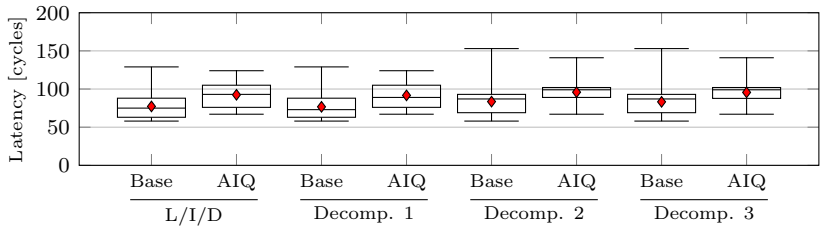


Figure 5.11.: Boxplot of observed latency of NoC transactions of ADA with 4 batches in a protected (AIQ) and unprotected NoC (Base).

- *Slv if.* for initiating NoC transactions;
- *IRQ if.* for interrupt forwarding.

Figures 5.12 and 5.13 details the resource usage (FPGA) and cell area (ASIC), respectively, of four different versions of the NI and their internal components: baseline (B) and different configurations of AIQ (2, 4, and 8) where AIQ is equipped with tracking tables with 2, 4 and 8 entries, respectively. In the figures, besides the four aforementioned interfaces, the NI is further divided into multiplexer and demultiplexer (*DE/MUX*), which connects the interfaces to the lower layers of the NoC; the lower layers of the NoC are captured by *Others*. The AIQ is subdivided into the core AIQ and the CRC integrity check (*CRC gen* and *CRC check*). In the ASIC, the results also show *Addr. Transl.*, which contains the NoC routes and VCs (source routing) as well as local and remote address mapping. In the FPGA, *Addr. Transl.* is captured by *Others*.

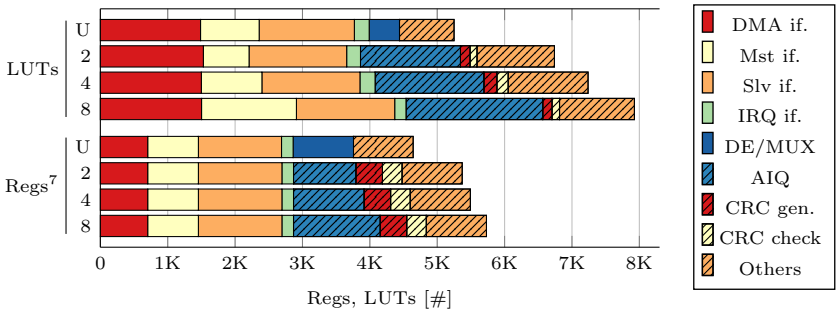


Figure 5.12.: FPGA synthesis results for a NI with AIQ (2, 4 and 8 entries) and without it (U).

In the FPGA (Figure 5.12), the total amount of data in the NI increases less than 2% with AIQ for all configurations. Optimized for FPGAs, the configuration memory and buffers are mapped to a total of seven Block RAMs: five with size 36Kb and two with size 16Kb. Notice that the number of BRAMs don't vary with the different configurations and, thus, they are not shown in the figure. The approach requires 15.7% additional register bits and 28.4% additional logic in the NI to implement AIQ with 2 table entries. The implementation overhead of AIQ is introduced by the main

[7]Not including the data buffers, synthesized as Block RAMs.

AIQ component and the CRC integrity check. The main component (*AIQ*) is responsible for most of the additional logic (19.7% additional LUTs), but requires only 0.6% additional register bits. Notice that the main component replaces the original multiplexer and demultiplexer (*DE/MUX*) as it already provides that functionality. The integrity check (*CRC gen.* and *CRC check*) is responsible for most of the additional register bits (14.8%), but requires only 4.7% additional logic (LUTs). When increasing the number of entries from 2 to 4, additional 2.3% register bits and 7.4% LUTs are required. Further doubling the number of entries from 4 to 8 requires additional 4.4% register bits and 9.5% LUTs. Notice, however, that some of that apparent overhead is not caused by AIQ. Some variations can be observed in the results that are caused by optimizations carried out by the toolchain – e.g., varying number of LUTs used by *Mst if* despite being independent of the number of entries of AIQ. Although FPGAs provide for fast prototyping and experimentation, ASIC is still the technology with which the work is expected to be applied.

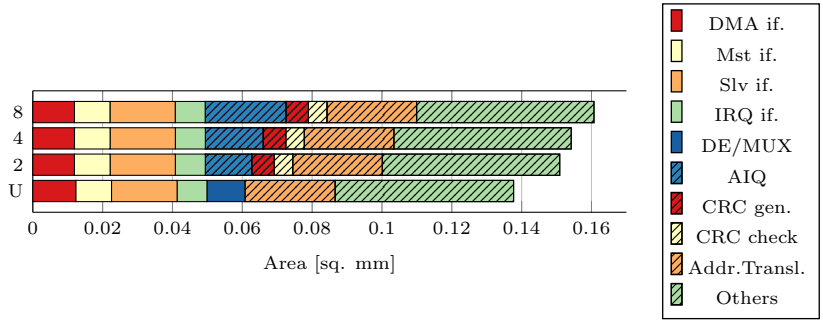


Figure 5.13.: ASIC synthesis results (65nm UMC) for a NI with AIQ (2, 4 and 8 entries) and without it (U).

In the 65nm UMC ASIC (Figure 5.13), a NI implementing AIQ with 2 table entries requires 9.6% additional silicon area (or NAND2-equivalent additional gates). The implementation overhead of AIQ is introduced by the main AIQ component and the CRC integrity check. In contrast to the FPGA, the main contributors to the area increase is the integrity check (*CRC gen.* and *CRC check*), which corresponds for approx. 82.9% of the additional logic. The main component (*AIQ*) contributes with only approx. 17.1% of the additional logic, since it replaces the original multiplexer and

demultiplexer (*DE/MUX*). When increasing the number of entries from 2 to 4, 2.2% additional logic is required (in total, 12.0% additional logic from U to 4). Further doubling the number of entries from 4 to 8 requires 4.2% additional logic (in total, 16.7% additional logic from U to 8). Nonetheless, both in the FPGA and in the ASIC, the resource and logic requirements of AIQ are expected to decrease with more efficient implementations of the NI and of AIQ itself.

Regarding maximum achievable frequency, the AIQ design does not impact negatively the circuit. On the contrary. On the FPGA, the baseline NI reached 78MHz whereas the version with AIQ reached just over 82MHz. That is due to the reduced critical path obtained when replacing the multiplexer and demultiplexer with AIQ, which split that pipeline stage in two. In the ASIC implementation, which reached 125MHz with 65nm UMC and 1GHz with 28nm TSMC, AIQ has no influence on the critical path. The critical path was found to lie in one of the interfaces in all configurations.

The energy consumption was evaluated with Synopsys PrimeTime [146] using the 65nm netlist. Under full load (single-word memory accesses with random payload), the NI equipped with AIQ (with 2 entries) consumes 8.03% more energy than baseline. For a larger AIQ with 4 and 8 entries the NI consumes 9.24% and 11.52% more energy than baseline, respectively. When idle (no traffic), the overheads are 6.45%, 7.70% and 9.91% for AIQ with 2, 4 and 8 table entries, respectiely.

The hardware cost of the AIQ approach is compared next with the resilient NoC approach, introduced in Chapter 3, and the DMR and TMR approaches [67]. DMR is able to detect errors of the NoC but it is neither able to pin-point nor to recover from them (cf. Section 1.5.2). As AIQ, DMR can be used to detect errors and achieve integrity. TMR is able to detect errors and also is able to detect which NoC instance is faulty. It can tolerate one error and continue operating. Error recovery is possible, e.g., by resetting the faulty NoC.

Before presenting the results, three considerations are required. First, the costs of DMR and TMR do not include the voter and recovery logic, and the Resilient NoC's NIs use AIQ as a lower bound for a full-featured ARQ implementation. Second, the results do not account for the link wires, which are highly dependent on the place and routing of the entire MPSoC. And third, for the FPGA, the synthesis tool optimally maps buffers and register banks to Block RAMs. The number of BRAMs is

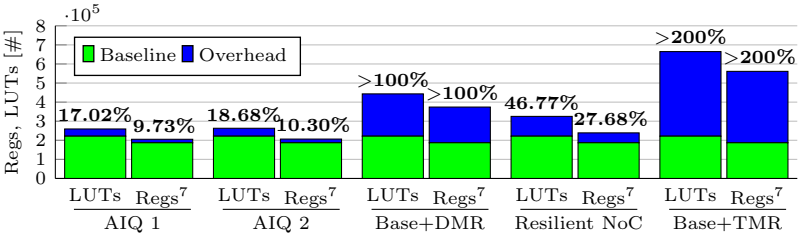


Figure 5.14.: Comparison of approach overhead for a 5x5 NoC (Virtex-7 FPGA).

not reported because it does not increase for the compared approaches except for DMR and TMR, which duplicates and triplicates the number of BRAMs, respectively.

Figure 5.14 shows the cost for FPGAs, in terms of resource usage, of a 5x5 NoC, where every router is connected to a NI. The total cost of implementing AIQ and ensuring the predictability and integrity of a 5x5 NoC is 17.02% in terms of logic and 9.73% in terms of registers when equipping the NIs with AIQ instances with 2 table entries (AIQ 1). When considering that two NIs are potential bottlenecks and require larger AIQ instances with 8 table entries (AIQ 2), the cost raises slightly to 18.68% in terms of logic and 10.30% in terms of registers. In contrast with DMR and its >100% overhead, AIQ requires just a fraction of the resources to provide the same guarantees. In order to further achieve resilience and high reliability, the resilient NoC approach requires 46.77% additional logic and 27.68% register bits. In comparison with AIQ, that is roughly three times the overhead. Notice that the resource requirement reported might exceed the ones available in the actual FPGA. Due to FPGA limitations, the reported figures are an extrapolation from the individual components required for implementing such a topology. As seen next, besides being the target technology for such a design, ASIC provides a better perspective with respect to the overheads.

Figure 5.15 shows the cost, in terms of area, for an ASIC implementation of the same scenario as above. In contrast with the FPGA, the cell area metric in ASIC concentrates all requirements in terms of sequential as well as combinational logic and provides, therefore, for a better overhead comparison. Moreover, the figure shows the results for two different

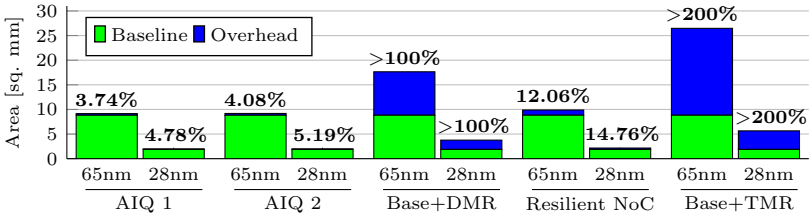


Figure 5.15.: Comparison of approach area overhead for a 5x5 NoC (65nm UMC and 28nm TSMC ASICs).

technology nodes: UMC's 65nm and TSMC's 28nm. Notice that this figure is intended to compare the overhead of different approaches in a given technology node. A comparison between the different technology nodes is provided in the sequel.

In a 65nm UMC ASIC, the total cost of implementing AIQ and ensuring the predictability and integrity of a 5x5 NoC is 3.74%, where NIs are equiped with AIQ instances with 2 table entries (AIQ 1). Considering the scenario with two larger AIQs (AIQ 2), the cost raises very slightly to 4.08%. Even when all NIs feature large AIQs, the cost corresponds to 6.52% (not plotted). In the considered setup, the overhead of DMR and TMR in ASIC does not differ from FPGA. When considering the resilient NoC approach, high reliability demands an area overhead of 12.06%. In comparison with AIQ, that is roughly three times the overhead. In comparison to TMR, not only is the resilient NoC much more efficient with respect to resource usage (FPGA) and area (ASIC) and, thus, power but also more effective. Besides, DMR and TMR imply a significant increase in the number of interconnecting wires, which can lead to circuit routing complications, potential congestion and lower frequencies.

In a 28nm TSMC ASIC (Figure 5.15), similar trends are observed albeit with a slightly larger overhead. That comes from the fact that different cells in a cell library – e.g., sequential, combinational, and their different variations – have different sizes. When scaling down the technology node from 65nm to 28nm, the corresponding cells in the library do not scale equally. That results in the case where, e.g., sequential cells in the smaller technology node are relatively larger than the combinatorial cells, in comparison with the larger technology node. Thus, although the synthesized

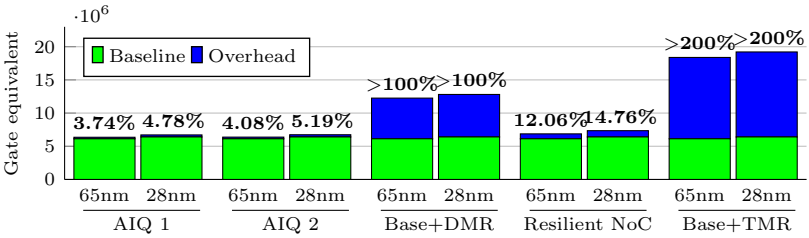


Figure 5.16.: Gate-equivalent comparison of approach overhead in different ASIC technology nodes (65nm UMC and 28nm TSMC) for a 5x5 NoC.

design is the same, the proportion between different cells (e.g., sequential and combinatorial) in the implementation results in a different area overheads. A comparison between the two technology nodes can be seen in Figure 5.16 with the gate equivalent metric. The gate equivalent aims at measuring the manufacturing-technology-independent complexity of a digital circuit [75]. The gate equivalent unit corresponds to the design silicon area divided by the area of a reference cell in the same technology node. A two-input drive-strength-one NAND cell (NAND2), which is usually used as reference, is employed here. The effects mentioned above can be observed here also for the baseline design. Notice that the baseline design requires relatively more silicon area (between 4% and 5%) in 28nm than in 65nm.

5.4.4. Reliability

To put into perspective what can be achieved with such overheads, the resulting reliability is also evaluated. The reliability is evaluated by means of the reliability metric $\mathcal{R}(t)$, which is the probability that the NoC does not fail during a time interval $[0, t]$ [67]. The definition of failure of Chapter 3 is reused, which states that failure in a high dependability mixed-critical real-time system is the violation of integrity, resilience or real-time latency guarantees due to errors, including static effects leading to blocking. Packet loss is not considered as a failure for and AIQ and the Resilient NoC because it is handled in the transport layer. The evaluation considers BERs

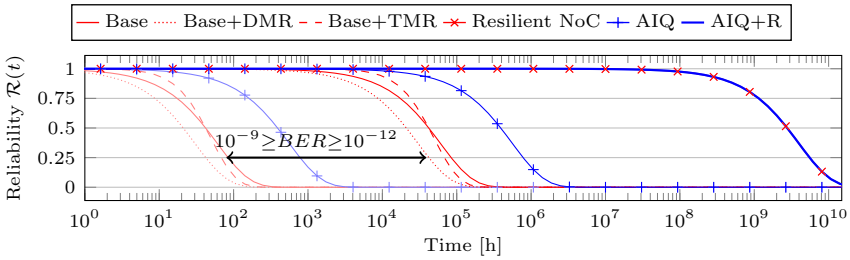


Figure 5.17.: Reliability comparison of the resilient NoC and non-resilient ones. A 5x5 NoC size is considered.

from 10^{-12} up to 10^{-9}/hour, which accounts for the expected[8] BERs in practice [6] (in the order of 10^{-12} bit-flips per hour) and the design safety margin [91] (up to 10^{-9}/hour). Additionally, a permanent fault rate[9] of 10^{-11}/h per router is considered. The occurrence of a permanent fault leads directly to failure.

Figure 5.17 plots the analytical $\mathcal{R}(t)$ for the unprotected, baseline NoC and the NoC protected with AIQ considering a 5x5 2D-mesh topology, excluding the NIs. The plot also includes the Resilient NoC and variants of the baseline NoC with DMR (Base+DMR) and TMR (Base+TMR). The DMR and TMR have ideal voters and the latter is non-reparable [67]. Although DMR can ensure the integrity of the system, the extra redundant hardware implies higher susceptibility to errors and results in a less reliable NoC than baseline. The same is seen for TMR, whose capability of withstanding one error does not pay off the extra redundant hardware. On the other hand, AIQ captures all violations of integrity and real-time requirements and is able to increase the reliability in about one order of magnitude w.r.t. the baseline NoC. However, there still exists the possibility that a soft error affects the state of the NoC and causes static effects, leading to the blocking of the NoC (cf. Chapter 2), which limits the reliability in time. That scenario can either be prevented with the Resilient NoC approach or it can be handled by resetting the NoC back to a valid state with a version of AIQ with NoC resetting capabilities (AIQ+R). AIQ+R and Resilient achieve equally high reliability, since all soft errors in the

[8] BERs derived for sequential and combinational logic [61] with data from [6] for 65nm CMOS SRAM. Masking effects [41] are not taken into account.

[9] The fault rate per router is inspired from processor failure rates in [111].

NoC can be detected and handled by both techniques, at which point hard errors become then the limiting factor.

Table 5.3.: Comparison of MTTFs in hours

	BER/h	3x3	5x5	8x8
Base	10^{-12}	$5.36 \cdot 10^5$	$6.07 \cdot 10^3$	$8.65 \cdot 10^3$
	10^{-9}	$5.36 \cdot 10^2$	$6.07 \cdot 10^1$	$8.65 \cdot 10^0$
Base+DMR	10^{-12}	$2.68 \cdot 10^5$	$3.03 \cdot 10^4$	$4.32 \cdot 10^3$
	10^{-9}	$2.68 \cdot 10^2$	$3.03 \cdot 10^1$	$4.32 \cdot 10^0$
Base+TMR	10^{-12}	$4.47 \cdot 10^5$	$5.06 \cdot 10^4$	$7.20 \cdot 10^3$
	10^{-9}	$4.47 \cdot 10^2$	$5.06 \cdot 10^1$	$7.20 \cdot 10^0$
AIQ	10^{-12}	$4.96 \cdot 10^6$	$5.63 \cdot 10^5$	$8.04 \cdot 10^4$
	10^{-9}	$4.96 \cdot 10^3$	$5.63 \cdot 10^2$	$8.04 \cdot 10^1$
AIQ+R & Resilient NoC	10^{-12}	$1.11 \cdot 10^{10}$	$4.00 \cdot 10^9$	$1.56 \cdot 10^9$
	10^{-9}	$1.11 \cdot 10^{10}$	$4.00 \cdot 10^9$	$1.56 \cdot 10^9$

The conclusions above can also be seen in the results of Table 5.3, which reports the MTTF in hours for different NoC sizes and BERs. Even in small topologies, non-resilient NoCs present very low MTTFs and consequently very high failure rates. On average, an 8x8 NoC is expected to be struck by a soft error from every 360 days in a regular environment (BER 10^{-12}) up to every 8.65 hours in an aggressive environment (BER 10^{-9}). Most of those errors will present only transient effects that will be handled in software. However, as mentioned above, some of them present static effects and their recovery requires resetting the NoC's state. Those are seldom and are expected to occur, on average, every 80.4 hours under BER 10^{-9}. The recovery requires cycles of downtime and thus impacts the NoC availability, which is evaluated next.

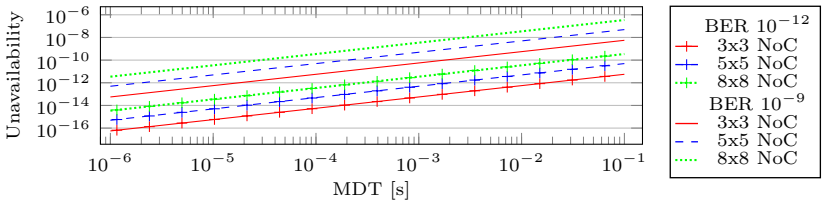


Figure 5.18.: Unavailability of the NoC with AIQ+R when varying the MDT.

Figure 5.18 reports the unavailability of the NoC, as the complement of its availability ($U = 1 - A$) for different sizes and BERs when varying the MDT. Even for very high BERs, with the MDT in the range of microseconds, the NoC still presents very high availability. As the MDT approaches a tenth of a second, the system experiences longer interruptions due to the NoC recovery, which is likely to compromise its timeliness and violate, already at design time, the real-time guarantees. Thus, fast software routines should be used with hardware support to ensure low MDTs and the applicability of the approach. Alternatively, in case the availability is too low due to a combination of long MDTs, high BERs and large NoC sizes, the Resilient NoC approach, which ensures the availability of the NoC in hardware, can be employed.

5.5. Summary

This chapter presented AIQ, an end-to-end mechanism to provide *integrity* and *real-time* guarantees for NoC transactions under random hardware faults. When integrated, the mechanism results in a low-overhead fault-tolerant NoC capable of detecting errors and ensuring that their effects are contained in time in order to maintain the system's *predictability* and *integrity*. AIQ explores the idea of keeping track of transactions of distributed systems in the context of NoCs for predictable real-time systems. Upon error detection, error handling and recovery are delegated to software, which may react according to an arbitrary strategy. The mechanism was evaluated in the many-core research platform IDAMC considering aspects such as performance and implementation costs. AIQ operates with high performance and low hardware overheads. The experimental evaluation with an industrial avionics use case shows an overhead between 1.4% and 7.1% in execution time. With respect to hardware overhead, AIQ requires less than 4.1% additional silicon area in an 65nm CMOS ASIC implementation of a 5x5 NoC. In a smaller technology node (28nm), AIQ requires less than 5.2% additional silicon area.

6. Replica-aware Co-scheduling for Mixed-criticality

Technology downscaling has increased the hardware's overall susceptibility to errors to the point where they became non-negligible. Hence, current and future computing systems must be appropriately designed to cope with errors in order to provide a reliable service and correct functionality. Specially in the real-time mixed-criticality domain, where applications with different requirements and criticalities co-exist in the system. The system must thus provide *sufficient independence* and prevent error propagation between criticalities [72, 35] – e.g., timing violation and data corruption. As previously seen in Chapter 2, depending on where and when errors occur, their impact on software execution range from masked (no observable effect) to a complete system crash. To handle such errors, the approaches can vary from completely software-based to completely hardware-based. The former are able to cover only part of the errors [45, 38] and the latter result in costly redundant hardware [64], as currently seen in lock-step dual-core execution [110]. This chapter addresses a more effective and efficient cross-layer approach that distributes the tasks of detecting errors and recovering from them in different layers of software and hardware [64, 45, 38].

A cross-layer fault-tolerance solution for mixed-criticality that increases the system's reliability at a higher level of abstraction without resorting to hardware redundancy has been developed in the ASTEROID project [10, 38]. ASTEROID's architecture is illustrated in Figure 6.1. The reliable software execution is realized by the operating system service Romain [38]. Mixed-critical applications may co-exist in the system and they are translated into protected and unprotected applications. Romain replicates the protected applications and manage their execution. Error detection is realized by a set o mechanisms whose main feature is the hardware assisted state comparison, which compares the replicas' state at certain points in time [10, 38]. Error recovery strategies can vary depending on whether the application is running in DMR or TMR [10, 9].

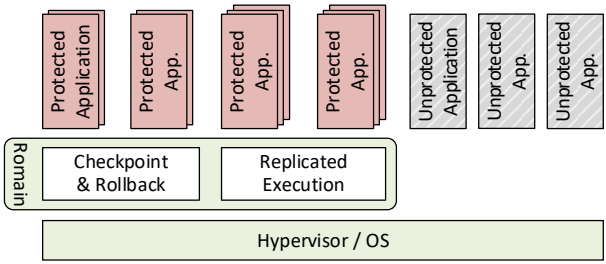


Figure 6.1.: ASTEROID's fault-tolerant architecture: the software side.

The performance of replicated execution has been analyzed by Axer et al. in [12] and revised in [8]. The work supports Partitioned Strict Priority Preemptive (SPP) scheduling where tasks are mapped to arbitrary cores, and assumes a single error model. The authors found that SPP, although widely employed in real-time systems, provides very pessimistic response time bounds for replicated tasks. Depending on the interfering workload, replicated tasks executing serially (on the same core) present in several cases better performance than when executing in parallel (on distinct cores). That occurs due to the long time that replicated tasks potentially have to wait on each core to synchronize and compare states before resuming execution. That leads to very low resource utilization and prevents the use of replicated execution in practice.

This chapter explores co-scheduling to provide small response times for replicated tasks without hindering the remaining unprotected tasks. Co-scheduling is a technique that schedules interacting tasks/threads to execute simultaneously on different cores [113]. It allows tasks/threads to communicate more efficiently by reducing the time they are blocked during synchronization. In contrast to SPP [12, 8], the proposed approach drastically minimizes delays due to the implicit synchronization found in state comparisons. In contrast to gang scheduling [48], it rules out starvation and distributes the execution of replicas in time to achieve small response times of unprotected tasks. Unlike standard Time-Division Multiplexing (TDM) and TDM with background partition [76], the approach ensures that all tasks have formal guarantees. Moreover, in contrast to related work, it supports different recovery strategies and accounts for the NoC communication delay and overheads due to replica management and state comparison.

The remaining of the chapter is organized as follows. A review of relevant related work is presented in Section 6.1. The replica-aware co-scheduling for mixed-criticality is introduced next, in two parts: the approach in Section 6.2 and the formal WCRT analysis in Section 6.3. The evaluation is presented in Section 6.4. Finally, Section 6.6 completes the chapter with a summary.

6.1. Related work

L4/Romain [38] is a cross-layer fault-tolerance approach that provides reliable software execution under soft errors. Romain provides protection at the application-level by replicating and managing the applications' executions as an operating system service. The error detection is realized by a set of mechanisms [10, 38, 45] whose main feature is the hardware assisted state comparison, which allows an effective and efficient comparison of the replicas' states. Pipeline fingerprinting [10] provides a checksum of the retired instructions and the pipeline's data path in every processor, detecting errors in the execution flow and data. The state comparison, which is reduced to comparing checksums instead of data structures, is carried out at certain points in time. It must occur at least when the application is about to externalize its state – e.g., in a *syscall* [38]. The replica generated *syscalls* are intercepted by Romain, have their integrity checked and their replicas' states compared before being allowed to externalize the state [38].

Mixed-criticality, in the context of this work, is supported with different levels of protection for applications with different criticalities and requirements (unprotected, protected with DMR[1]or TMR) and by ensuring that timing constraints are met even in case of errors. For instance, Romain provides different error recovery strategies [10, 9]:

- *DMR with checkpoint and rollback*: to recover, the replicas rollback to their last valid state and re-execute;
- *TMR with state copy*: to recover, the state of the faulty replica is replaced with the state of one of the healthy replicas.

This chapter focuses on the system-level timing aspect of errors affecting the applications. It assumes thereby the absence of failures in critical components [45, 126], such as the OS, the replica manager/voter (e.g., Romain) and the interconnect (e.g., NoC), which can be protected as in

Chapters 3 and 5 and [66, 132].

The WCRT of replicated execution has been analyzed by Axer et al. in [12], where replicas are modeled as fork-join tasks in a system implementing Partitioned SPP. The work was later revised in [8] due to optimism in the original approach. The revised approach is the version considered in this thesis. In the approach, with deadline monotonic priority assignment, where the priority of tasks decrease as their deadlines increase, replicated tasks often perform worse when mapped in parallel than when mapped to a single core. That is due to the state comparisons during execution, which involves implicit synchronization between cores. With partitioned scheduling, in the worst-case, the synchronization ends up accumulating the interference from all cores to which the replicated task is mapped, resulting in poor performance in higher loads. On the other hand, mapping replicated tasks to the highest priorities results in long response times for lower priority tasks and rules out deadline monotonicity. The latter causes the unschedulability of all tasksets with at least one regular task whose deadline is shorter than the execution time of a replicated task.

Gang scheduling [48] is a co-scheduling variant that schedules groups of interacting tasks/threads simultaneously. It increases performance by reducing the inter-thread communication latency. The authors in [78] present an integration between gang scheduling and Global Earliest Deadline First (EDF), called the Gang EDF. They provide a schedulability analysis derived from the Global EDF's based on the sporadic task model. In another work, [53] shows that SPP Gang schedulers are in general not predictable, for instance, due to priority inversions and slack utilization. In the context of real-time systems, gang scheduling has not received much attention.

TDM-based scheduling [76] is widely employed to achieve predictability and ensure temporal-isolation. Tasks are allocated to partitions, which are scheduled to execute in time slots. Partitions can span across several (or all) cores and can be executed at the same time. The downside of TDM is that it is not work-conserving and underutilizes system resources. A TDM variant with background partition [76] tackles that issue by allowing low priority tasks to execute in other partitions whenever no higher priority workload is executing. Yet, in addition to the high cost to switch between

[1] DMR *per se* can be used for system integrity only. However, DMR augmented with checkpointing and rollback enables recovery and can be used to achieve integrity and availability (state rollback followed by re-execution in both replicas) [10, 9].

partitions, no guarantees can be given to tasks in the background partition.

The proposed replica-aware co-scheduling for mixed-criticality exploits co-scheduling with SPP to improve the performance of the system. The approach differs from [12, 8] in that replicas are treated as gangs and are mapped with highest priorities, and are hence activated simultaneously on different cores. In contrast to gang-scheduling [48, 53] and to [12, 8], the execution of replicas is distributed in time with offsets to compensate for the lack of deadline monotonicity thus allowing the schedulability of tasks with short deadlines. The approach further provides for the worst-case performance of lower priority tasks by allowing them to execute whenever no higher priority workload is executing. However, in contrast to [76], all tasks have WCRT guarantees. Finally, the analysis also models the state comparison and the on-Chip communication overheads, and although Romain is used as an example, it is also applicable to other approaches.

6.2. The proposed approach

This section introduces the replica-aware co-scheduling for mixed-criticality including its assumptions, system, task, and event models. The modeling is realized with CPA [62] to provide formal response time bounds.

6.2.1. System model

The system consists of a standard NoC-based many-core composed of processing elements, simply referred to as cores.

There are two types of tasks in the system, as in [8]:

- *independent* tasks τ_i: regular, unprotected tasks; and
- *fork-join* tasks Γ_i: replicated, protected tasks.

The system implements partitioned scheduling, where the operating system manages tasks statically mapped to cores. The mapping is assumed to be given as input. The scheduling policy is a combination of SPP and gang scheduling. When executing only independent tasks, the system's behavior is identical to Partitioned SPP, where tasks are scheduled independently on each core according to SPP. It differs from SPP, when scheduling fork-join tasks.

Fork-join tasks are mapped with highest priorities, hence do not suffer

interference from independent tasks, and execute simultaneously on different cores, as in gang scheduling. Note that deadline monotonicity is therefore only partially possible. To limit the interference to independent tasks, the execution of a fork-join task is divided in smaller intervals called stages, whose executions are distributed in time. At the end of each stage, the states of the replicas are compared. In case of an error, i.e., states differ, recovery is triggered.

Fork-join stages are executed with static offsets [115] in execution slots. One stage is executed per slot. On a core with n fork-join tasks, there are $n + 1$ execution slots: one slot for each fork-join task Γ_i and one slot for recovery. The slots are cyclically scheduled in a cycle Φ. The slot for Γ_i starts at offset $\phi(\Gamma_i)$ relative to the start of Φ and ends after $\zeta(\Gamma_i)$, the slot length. The recovery slot is shared by all fork-join tasks on that core and is where error recovery may take place under a single error assumption (details in Sections 6.2.3 and 6.3.3). The recovery slot has an offset $\phi(recovery)$ relative to Φ and length $\zeta(recovery)$. Lower priority independent tasks are allowed to execute whenever no higher priority workload is executing.

An example is shown in Figure 6.2, where two fork-join tasks Γ_1 and Γ_2 and two independent tasks τ_3 and τ_4 are mapped to two cores. Γ_1 and Γ_2 execute in their respective slots simultaneously in both cores. When an error occurs, the recovery of Γ_2 is scheduled and the recovery of the error-affected stage occurs in the recovery slot. The use of offsets enables the schedulability of independent tasks with short periods and deadlines, such as τ_3 and τ_4. Note that, without the offsets, Γ_1 and Γ_2 would execute back-to-back leading to the unschedulability of τ_3 and τ_4.

6.2.2. Task model

An independent task τ_i is mapped to core σ with a priority p. Once activated, it executes for at most C_i, its WCET. The activations of a task are modeled with arbitrary *event models*. Task activations in an event model are given by arrival curves $\eta^-(\Delta t)$ and $\eta^+(\Delta t)$, which return the minimum and maximum number of events arriving in any time interval Δt. Their pseudo-inverse counterparts $\delta^+(q)$ and $\delta^-(q)$ return the maximum and minimum time interval between the first and last events in any sequence of q event arrivals. Conversion is provided in [135]. Periodic events with jitter, sporadic events and others can be modeled with the minimum

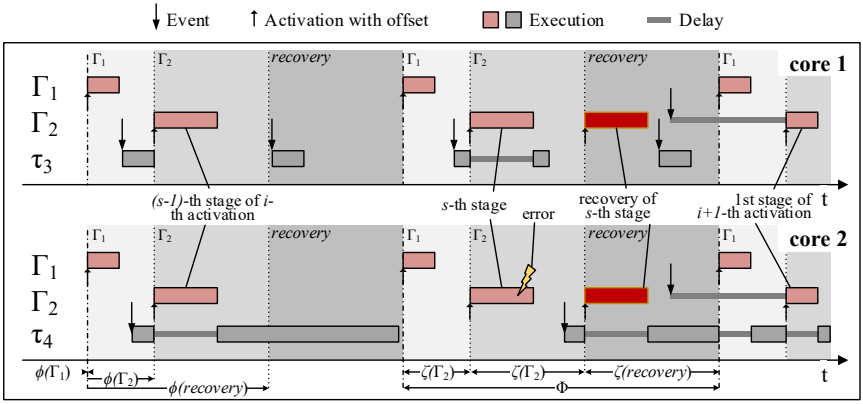


Figure 6.2.: Execution example with two fork-join and two independent tasks on two cores.

distance function $\delta_i^-(q)$ as follows [135]:

$$\delta_i^-(q) = \max((q-1)\cdot d^{min}, (q-1)\cdot \mathcal{P} - \mathcal{J}) \tag{6.1}$$

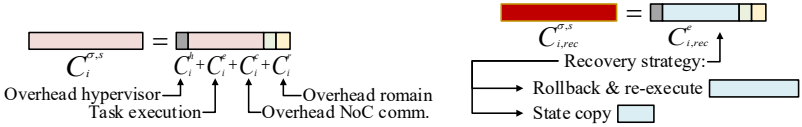
where $\mathcal{P}$ is the period, $\mathcal{J}$ is the jitter, d^{min} is the minimum distance between any two events, and the subscript i indicates the association with a task τ_i or Γ_i.

Fork-join tasks are rigid parallel tasks, i.e., the number of processors required by a fork-join task is fixed and specified externally to the scheduler [53], and consist of multiple stages with data dependencies, as in [8, 3]. A fork-join task Γ_i is a Directed Acyclic Graph (DAG) $G(V, E)$ where vertices in V are subtasks and edges in E are precedence dependencies [8]. In the graph, tasks are partitioned in *segments* and *stages*, as illustrated in Figure 6.4a. A subtask $\tau_i^{\sigma,s}$ is the s-th stage of the σ-th segment and is annotated with its WCET $C_i^{\sigma,s}$. The WCET of a stage is equal across all segments, i.e., $\forall x, y \colon C_i^{x,s} = C_i^{y,s}$. Each segment σ of Γ_i is mapped to a distinct core. A fork-join task Γ_i is annotated with the *static offset* $\phi(\Gamma_i)$, which marks the start of its execution slot in Φ. The offset also admits a small positive jitter j_ϕ, to account for a slight desynchronization between cores and context switch overhead.

The activations of a fork-join task are modeled with *event models*. Once Γ_i is activated, its stages are successively activated by the completion of all

segments of the previous stage, as in [8, 3]. The proposed approach differs from them in that it restricts the scheduling of at most one stage of Γ_i in a cycle Φ, and the stage receives service at the offset $\phi(\Gamma_i)$. Note that the event arrival at a fork-join task is not synchronized with its offset. The events at a fork-join task are queued at the first stage and only one event at a time is processed (FIFO) [8]. A queued event is admitted when the previous event leaves the last stage.

The interaction with Romain (the voter) is modeled in the analysis as part of the WCET $C_i^{\sigma,s}$, as depicted in Figure 6.3a. The WCET includes the on-Chip communication latency and state comparison overheads, as the Romain instance may be mapped to an arbitrary core. Those can be obtained, e.g., with [130] along with task mapping and scheduler properties to avoid over-conservative interference estimation and obtain tighter bounds.



(a) WCET of a fork-join subtask (b) WCET of recovery

Figure 6.3.: The composition of WCET of fork-join subtasks.

6.2.3. Error model

The error model assumes a single error scenario caused by SEEs (cf. Section 2). It assumes that all errors affecting fork-join tasks (replicated, protected tasks) can be detected and contained, ensuring integrity. The overhead of error detection mechanisms are modeled as part of the WCET (cf. Figure 6.3a). Regarding independent tasks (regular, unprotected tasks), it assumes that an error immediately leads to a task failure: an independent task has no integrity guarantee, no explicit error detection and no error recovery. It is assumed that the failure of an independent task will not violate the integrity and the WCRT guarantees of the remaining tasks. Those assumptions are met, e.g., by Romain[2]with mechanisms such as WCET monitoring to contain execution overruns. Moreover, it assumes the absence of failures in critical components [45, 126], such as the OS, the replica

manager/voter Romain and the interconnect (e.g., the NoC), which can be protected as in Chapters 3 and 5 and [66, 132].

The model provides recovery[2] for fork-join tasks, ensuring their availability. With a recovery slot in every cycle Φ, the proposed approach is able to handle up to one error per cycle Φ. However, the analysis in Section 6.3.3 assumes at most one error per busy window (the concept will be introduced in Section 6.3). The assumption is reasonable since the probability of a multiple error scenario in a busy window, which is usually shorter than a second in practice, is very low and can be considered as an acceptable risk [72]. A multiple error scenario occurs only if an error affects more than one replica at a time or if more than one error occurs within the same busy window.

6.2.4. Offsets

The execution of fork-join tasks in the proposed approach is based on static offsets, which are assumed to be provided as input to the scheduler. The offsets form execution slots whose sizes do not vary during runtime, as seen in Figure 6.2. Varying the slots' sizes would substantially increase the timing analysis complexity without a justifiable performance gain. The offsets must satisfy two constraints:

Constraint 6.1. *A slot for a fork-join task Γ_i must be large enough to fit the largest stage of Γ_i. That is, $\forall s, \sigma$: $\zeta(\Gamma_i) \geq C_i^{\sigma,s} + j_\phi$.*

Constraint 6.2. *The recovery slot must be large enough to fit the recovery of the largest stage of any fork-join task mapped to that core. That is, $\forall i, s, \sigma$: $\zeta(recovery) \geq C_{i,rec}^{\sigma,s} + j_\phi$.*

where a one error scenario per cycle is assumed and $C_{i,rec}^{\sigma,s}$ is the recovery WCET of subtask $\tau_i^{\sigma,s}$ (cf. Section 6.3.3).

Next, basic offsets that satisfy Constraints 6.1 and 6.2 are provided. The calculation must consider only overlapping fork-join tasks, i.e., fork-join tasks mapped to at least one core in common. Offsets for non-overlapping fork-join tasks are computed separately as they do not interfere directly with each other. The indirect interference, e.g., in the NoC, are accounted

[2]Romain is able to detect and recover from all soft errors affecting user-level applications. For details on the different error impacts and detection strategies, the interested reader can refer to [10, 38].

for in the WCETs. First, one must find the smallest slots that satisfy Constraint 6.1:

$$\forall \Gamma_i : \zeta(\Gamma_i) = \max_{\forall \sigma, s} \{C_i^{\sigma,s}\} + j_\phi \tag{6.2}$$

and the smallest recovery slot that satisfies Constraint 6.2:

$$\zeta(recovery) = \max_{\forall \Gamma_i, \tau_j^{\sigma,s} \in \Gamma_i} \left\{C_{i,rec}^{\sigma,s}\right\} + j_\phi \tag{6.3}$$

The cycle Φ is then the sum of all slots:

$$\Phi = \sum_{\forall \Gamma_i} \{\zeta(\Gamma_i)\} + \zeta(recovery) \tag{6.4}$$

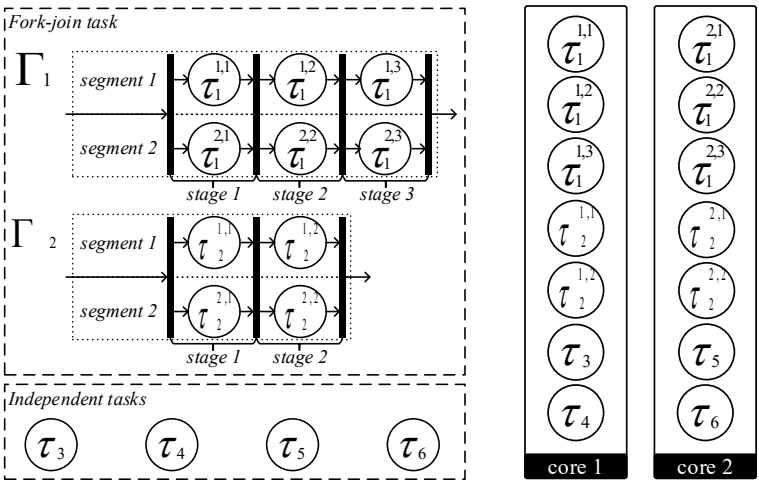
The offsets depend then on the order in which the slots are placed inside Φ. Assuming that the slots $\phi(\Gamma_i)$ are sorted in ascending order on i and that the recovery slot is the last one, the offsets are obtained by:

$$\phi(x) = \begin{cases} 0 & \text{if } x = \Gamma_1 \\ \phi(\Gamma_{i-1}) + \zeta(\Gamma_{i-1}) & \text{if } x = \Gamma_i \text{ and } i > 1 \\ \Phi - \zeta(recovery) & \text{if } x = recovery \end{cases} \tag{6.5}$$

6.3. Formal response-time analysis

The analysis is based on CPA and inspired by [8, 115]. In CPA, the WCRT is calculated with the busy window approach [149]. The response time of an event of a task τ_i (resp. Γ_i) is the time interval between the event arrival and the completion of its execution. In the busy window approach [149], the event with the WCRT can be found inside the busy window. The busy window w_i of a task τ_i (resp. Γ_i) is the time interval where all response times of the task depend on the execution of at least one previous event in the same busy window, except for the task's first event. The busy window starts at a critical instant corresponding to the worst-case scheduling scenario. Since the worst-case scheduling scenario depends on the type of task, it will be derived individually in the sequel.

Before the analysis for fork-join and for independent tasks is derived, Figure 6.4 introduces an example used throughout the section. The taskset consists of 4 independent tasks and 2 fork-join tasks, mapped to two cores. The task priority on each core decreases from top to bottom – e.g., $\tau_1^{1,1}$ has the highest priority and τ_4 the lowest.



(a) Taskset (b) Mapping

Figure 6.4.: A taskset with 4 independent tasks and 2 fork-join tasks, and its mapping to 2 cores. Highest priority at the top, lowest at the bottom.

6.3.1. Fork-join tasks

Deriving the WCRT for an arbitrary fork-join task Γ_i requires identifying the critical instant leading to the worst-case scheduling scenario. In case of SPP, the critical instant is when all tasks are activated at the same time and the tasks' subsequent events arrive as early as possible [149]. In this case, the critical instant must also account for the use of static offsets [115].

The worst-case scheduling scenario for Γ_2 on core 1 is illustrated in Figure 6.5. Γ_2 is activated and executed at the same time on cores 1 and 2 (omitted). Notice that, by design, fork-join tasks do not dynamically interfere with each other. The *critical instant* occurs when the first event of Γ_2 arrives just after missing Γ_2's offset. The event has to wait until the next cycle to be served, which takes time $\Phi + j_\phi$ when the activation with offset is delayed by a jitter j_ϕ. Notice that the WCETs of fork-join tasks already account for the inter-core communication and synchronization overhead (cf. Figure 6.3a).

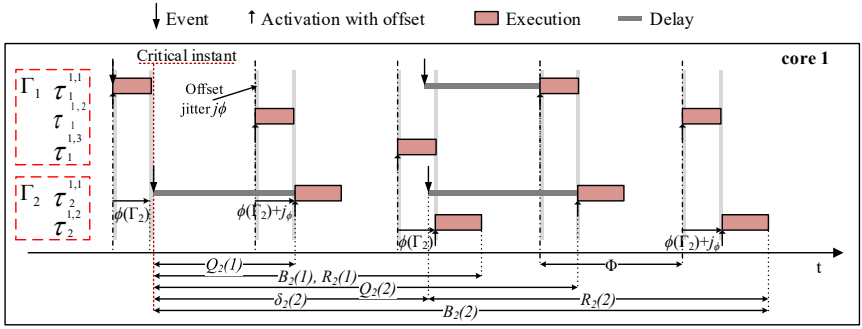


Figure 6.5.: Worst-case schedule for fork-join gang Γ_2 on core 1 (cf. Figure 6.4).

Lemma 6.1. *The critical instant leading to the worst-case scheduling scenario of a fork-join task Γ_i is when the first event of Γ_i arrives just after missing Γ_i's offset $\phi(\Gamma_i)$.*

Proof. A fork-join task Γ_i does not suffer interference from independent tasks or other fork-join tasks. The former holds since independent tasks always have lower priority. The latter holds due to three reasons: an arbitrary fork-join task Γ_j always receives service in its slot $\phi(\Gamma_j)$; the slot $\phi(\Gamma_j)$ is large enough to fit Γ_j's largest subtask (Constraint 6.1); and the slots in a cycle Φ are disjoint. Thus, the critical instant can only be influenced by Γ_i itself.

The proof is by contradiction. Suppose that there is another scenario worse than Lemma 6.1. That means that the first event can arrive at a time that causes a delay to Γ_i larger than $\Phi + j_\phi$. However, if the delay is larger than $\Phi + j_\phi$, then the event arrived before a previous slot $\phi(\Gamma_i)$ and Γ_i did not receive service. Since that can only happen if there is a pending activation of Γ_i and thus violates the definition of a busy window, the hypothesis must be rejected. □

The Multiple-Event Queueing Delay $Q_i(q)$ and Multiple-Event Busy Time $B_i(q)$, on which the busy window relies, are derived next. $Q_i(q)$ is the longest time interval between the arrival of Γ_i's first activation and the first time its q-th activation receives service, considering that all events belong to the same busy window [8, 89]. For Γ_i, the q-th activation can

receive service at the next cycle Φ after the execution of $q-1$ activations of Γ_i lasting $s_i \cdot \Phi$ each, a delay Φ (cf. Lemma 6.1) and a jitter j_ϕ. This is given by:

$$Q_i(q) = (q-1) \cdot s_i \cdot \Phi + \Phi + j_\phi \tag{6.6}$$

where s_i is the number of stages of Γ_i and Φ is the cycle.

Lemma 6.2. *The Multiple-Event Queueing Delay $Q_i(q)$ given by Equation 6.6 is an upper bound.*

Proof. The proof is by induction. When $q=1$, Γ_i has to wait for service at most until the next cycle Φ plus an offset jitter j_ϕ to get service for its first stage, considering that the event arrives just after its offset (Lemma 6.1). In a subsequent $q+1$-th activation in the same busy window, Equation 6.6 must also consider q entire executions of Γ_i. Since Γ_i has s_i stages and only one stage can be activated and executed per cycle Φ, it takes additional $s_i \cdot \Phi$ for each activation of Γ_i, resulting in Equation 6.6. □

The Multiple-Event Busy Time $B_i(q)$ is the longest time interval between the arrival of Γ_i's first activation and the completion of its q-th activation, considering that all events belong to the same busy window [8, 89]. The q-th activation of Γ_i completes after a delay Φ (cf. Lemma 6.1), a jitter j_ϕ and the execution of q activations of Γ_i. That is given by:

$$B_i(q) = q \cdot s_i \cdot \Phi + j_\phi + C_i^{\sigma,s} \tag{6.7}$$

where $C_i^{\sigma,s}$ is the WCET of Γ_i's last stage.

Lemma 6.3. *The Multiple-Event Busy Time $B_i(q)$ given by Equation 6.7 is an upper bound.*

Proof. The proof is by induction. When $q=1$, Γ_i has to wait for service at most until the next cycle Φ plus an offset jitter j_ϕ to get service for its first stage (Lemma 6.1), plus the completion of the last stage of the activation lasting $(s_i-1) \cdot \Phi + C_i^{\sigma,s}$. This is given by:

$$\begin{aligned} B_i(1) &= (s_i - 1) \cdot \Phi + \Phi + j_\phi + C_i^{\sigma,s} \\ &= s_i \cdot \Phi + j_\phi + C_i^{\sigma,s} \end{aligned} \tag{6.8}$$

In a subsequent $q+1$-th activation in the same busy window, Equation 6.7 must consider q additional executions of Γ_i. Since Γ_i has s_i stages and only one stage can be activated and executed per cycle Φ, it takes additional $s_i \cdot \Phi$ for each activation of Γ_i. Thus, Equation 6.7. □

Now the busy window and WCRT of Γ_i can be calculated. The busy window w_i of a fork-join task Γ_i is given by:

$$w_i = \max_{q \geq 1,\, q \in \mathbb{N}} \{B_i(q) \,|\, Q_i(q+1) \geq \delta_i^-(q+1)\} \tag{6.9}$$

Lemma 6.4. *The busy window is upper bounded by Equation 6.9.*

Proof. The proof is by contradiction. Suppose there is a busy window $\breve{w}_i$ longer than w_i. In that case, $\breve{w}_i$ must contain at least one activation more than w_i, i.e. $\breve{q} \geq q+1$. From Equation 6.9, we have that $Q_i(\breve{q}) < \delta_i^-(\breve{q})$, i.e. $\breve{q}$ is not delayed by the previous activation. Since that violates the definition of a busy window, the hypothesis must be rejected. □

The response time $R_i(q)$ of the q-th activation of Γ_i in the busy window is given by:

$$R_i(q) = B_i(q) - \delta_i^-(q) \tag{6.10}$$

The worst-case response time R_i^+ is the longest response time of any activation of Γ_i observed in the busy window.

$$R_i^+ = \max_{1 \leq q \leq \eta_i^+(w_i)} R_i(q) \tag{6.11}$$

Theorem 6.1. *R_i^+ (Equation 6.11) provides an upper bound on the worst-case response time of an arbitrary fork-join task Γ_i.*

Proof. The WCRT of a fork-join task Γ_i is obtained with the busy window approach [149]. It remains to prove that the critical instant leads to the worst-case scheduling scenario, that the interference captured in Equations. 6.6 and 6.7 are upper bounds, and that the busy window is correctly captured by Equation 6.9. These are proved in Lemmas 6.1, 6.2, 6.3, and 6.4, respectively. □

6.3.2. Independent tasks

Deriving the WCRT analysis of an arbitrary independent task τ_i requires identifying two types of interference that affect them: interference caused by higher priority independent tasks and by fork-join tasks. First, the critical instant leading to the worst-case scheduling scenario where τ_i suffers the most interference must be identified.

Lemma 6.5. *The critical instant of τ_i is when the first event of higher priority independent tasks arrive simultaneously with τ_i's event at the offset of a fork-join task.*

Proof. The worst-case interference caused by a higher priority (independent) task τ_j under SPP is when its first event arrives simultaneously with τ_i's and continue arriving as early as possible [149].

The interference caused by a fork-join task Γ_j on τ_i depends on Γ_j's offset $\phi(\Gamma_j)$ and subtasks $\tau_j^{\sigma,s}$, whose execution times vary for different stages s. Assume a critical instant that occurs at a time other than at the offset $\phi(\Gamma_j)$. Since a task Γ_j starts receiving service at its offset, an event of τ_i arriving at time $t > \phi(\Gamma_j)$ can only suffer less interference from Γ_j's subtask than when arriving at $t = 0$. □

Fork-join subtasks have different execution times for different stages, which leads to a number of scheduling scenarios that must be evaluated [115]. Each scenario is defined by the fork-join subtasks that will receive service in the cycle Φ and the offset at which the critical instant supposedly occurs. The scenario is called a critical instant candidate S. Since independent tasks participate in all critical instant candidates, they are omitted in S for the sake of simplicity.

Definition 6.1. *Critical Instant Candidate S: the critical instant candidate S is an ordered pair (a, b) where a is a critical offset and b is a tuple containing one subtask $\tau_j^{\sigma,s}$ of every interfering fork-join task Γ_j.*

The set of candidates that must be evaluated can then be defined.

Definition 6.2. *Critical Instant Candidate Set $\mathcal{S}$: the set containing all possible different critical instant candidates S.*

The worst-case schedule of the independent task τ_4 from the example in Figure 6.4 is illustrated in Figure 6.6. In fact, the critical instant leading to τ_4's WCRT is at $\phi(\Gamma_1)$ when $\tau_1^{1,2}$ and $\tau_2^{1,1}$ receive service at the same cycle Φ, i.e. $S = (\phi(\Gamma_1), (\tau_1^{1,2}, \tau_2^{1,1}))$. Events of the independent task τ_3 start arriving at the critical instant and continue arriving as early as possible.

The interference $I_i^I(\Delta t)$ caused by equal or higher priority independent tasks in any time interval Δt can now be bounded. The interference $I_i^I(\Delta t)$

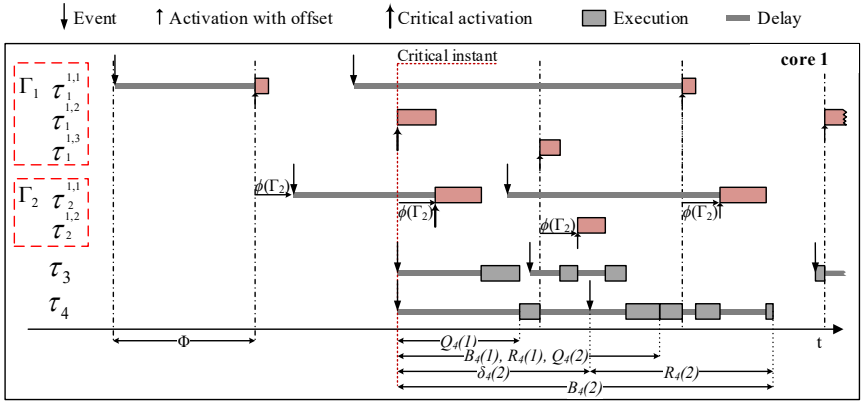


Figure 6.6.: The worst-case schedule for independent task τ_4 on core 1 (cf. Figure 6.4).

can be upper bounded as follows [89]:

$$I_i^I(\Delta t) = \sum_{\forall \tau_j \in hp_I(i)} \eta_j^+(\Delta t) \cdot C_j \tag{6.12}$$

where $hp_I(i)$ is the set of equal or higher priority independent tasks mapped to the same core as τ_i.

To derive the interference caused by fork-join tasks the Critical Instant Event Model must be defined. The critical instant event model $\check{\eta}_i^{\sigma,s}(\Delta t, S)$ of a subtask $\tau_i^{\sigma,s} \in \Gamma_i$ returns the maximum number of activations observable in any time interval Δt, assuming the critical instant S. It can be derived from Γ_i's input event model $\eta_i^+(\Delta t)$ as follows:

$$\check{\eta}_i^{\sigma,s}(\Delta t, S) = \min\left\{\eta_i^+\left(\Delta t_S + \Phi - \phi(\Gamma_i)\right), \psi\right\} - gt\left(s^S, s, \phi^S, \phi(\Gamma_i)\right) \tag{6.13}$$

$$\psi = \left\lfloor \frac{\Delta t_S}{\Phi \cdot s_i} \right\rfloor + ge\left(\Delta t_S \bmod (\Phi \cdot s_i),\ \Phi \cdot (s-1)\right) \tag{6.14}$$

$$\Delta t_S = \Delta t + \underbrace{\Phi \cdot (s^S - 1)}_{\substack{\text{critical instant} \\ \text{stage}}} + \underbrace{\phi^S}_{\substack{\text{critical instant} \\ \text{offset}}} \tag{6.15}$$

where s is the stage of subtask $\tau_i^{\sigma,s}$; s_i is the number of stages in Γ_i; ϕ^S is the offset in S; s^S is the stage of Γ_i in S; $gt(a,b,c,d)$ is a function that

returns 1 when $(a > b) \vee (a = b \wedge c > d)$, 0 otherwise; and $ge(a, b)$ is a function that returns 1 when $a \geq b$, 0 otherwise.

Lemma 6.6. *$\check{\eta}_i^{\sigma,s}(\Delta t, S)$ (Equation 6.13) provides a valid upper bound on the number of activations of $\tau_i^{\sigma,s}$ observable in any time interval Δt, assuming the critical instant S.*

Proof. The proof is by induction, in two parts.

First let us assume $s^S = 1$ and $\phi^S = 0$, neutral values resulting in $\Delta t_S = \Delta t$ and $gt(s^S, s, \phi^S, \phi(\Gamma_i)) = 0$. The maximum number of activations of $\tau_i^{\sigma,s}$ seen in the interval Δt is limited by the maximum number of activations of the fork-join task Γ_i because a subtask $\tau_i^{\sigma,s}$ is activated once per Γ_i's activation, and limited by the maximum number of times that $\tau_i^{\sigma,s}$ can actually be scheduled and served in Δt. That is ensured in Equation 6.13 by the minimum function and its first and second terms, respectively.

When $s^S > 1$ and/or $\phi^S > 0$, the time interval $[0, \Delta t)$ must be moved forward so that it starts at stage s^S and offset ϕ^S. That is captured by Δt_S in Equation 6.15 and by the last term of Equation 6.13. The former extends the end of the time interval by the time it takes to reach the stage s^S and the offset ϕ^S, i.e., $[0, \Delta t_S)$. The latter pushes the start of the interval forward by subtracting an activation of $\tau_i^{\sigma,s}$ if it occurs before the stage s^S and the offset ϕ^S, resulting in the interval $[\Delta t_S - \Delta t, \Delta t_S)$. Thus Equation 6.13. □

The interference $I_i^{FJ}(\Delta t, S)$ caused by fork-join tasks on the same core in any time interval Δt, assuming a critical instant candidate S, can then be upper bounded as follows:

$$I_i^{FJ}(\Delta t, S) = \sum_{\forall \tau_j^{\sigma,s} \in hp_{FJ}(i)} \check{\eta}_j^{\sigma,s}(\Delta t, S) \cdot C_j^{\sigma,s} \tag{6.16}$$

where $hp_{FJ}(i)$ is the set of fork-join subtasks mapped to the same core as τ_i.

The Multiple-Event Queueing Delay $Q_i(q, S)$ and the Multiple-Event Busy Time $B_i(q, S)$ for an independent task τ_i, assuming a critical instant candidate S, can be derived as follows.

$$Q_i(q, S) = (q - 1) \cdot C_i + I_i^I(Q_i(q, S)) + I_i^{FJ}(Q_i(q, S), S) \tag{6.17}$$

$$B_i(q, S) = q \cdot C_i + I_i^I(B_i(q, S)) + I_i^{FJ}(B_i(q, S), S) \tag{6.18}$$

where $q \cdot C_i$ is the time required to execute q activations of task τ_i.

Equations 6.17 and 6.18 result in fixed-point problems, similar to the well known busy window equation (Equation 6.9). They can be solved iteratively, starting with a very small, positive ϵ.

Lemma 6.7. *The Multiple-Event Queueing Delay $Q_i(q, S)$ given by Equation 6.17 is an upper bound, assuming the critical instant S.*

Proof. The proof is by induction. When $q = 1$, τ_i has to wait for service until the interfering workload is served. The interfering workload is given by Equations 6.12 and 6.16. Since $\eta_j^+(\Delta t)$ and C_j are upper bounds by definition, Equation 6.12 is also an upper bound. Similarly, since $\breve{\eta}_j^{\sigma,s}(\Delta t, S)$ is an upper bound (cf. Lemma 6.6) and $C_j^{\sigma,s}$ is an upper bound by definition, 6.16 is an upper bound for a given S. Therefore, $Q_i(1, S)$ is also an upper bound, for a given S.

In a subsequent $q + 1$-th activation in the same busy window, $Q_i(q, S)$ also must consider q executions of τ_i. That is captured in Equation 6.17 by the first term, which is, by definition, an upper bound on the execution time. From that, Lemma 6.7 follows. □

Lemma 6.8. *The Multiple-Event Busy Time $B_i(q, S)$ given by Equation 6.18 is an upper bound, assuming the critical instant S.*

Proof. The proof is similar to Lemma 6.7, except that $B_i(q, S)$ in Equation 6.18 also captures the completion of the q-th activation. It takes additional C_i, which is an upper bound by definition. Thus Equation 6.18 is an upper bound, for a given S. □

The busy window $w_i(q, S)$ of an independent task τ_i is given by:

$$w_i(S) = \max_{q \geq 1,\, q \in \mathbb{N}} \{B_i(q, S) \mid Q_i(q+1, S) \geq \delta_i^-(q+1)\} \tag{6.19}$$

Lemma 6.9. *The busy window is upper bounded by Equation 6.19.*

Proof. The proof is by contradiction. Suppose there is a busy window $\breve{w}_i(S)$ longer than $w_i(S)$. In that case, $\breve{w}_i(S)$ must contain at least one activation more than $w_i(S)$, i.e., $\breve{q} \geq q + 1$. From Equation 6.19, we have that $Q_i(\breve{q}, S) < \delta_i^-(\breve{q})$, i.e., $\breve{q}$ is not delayed by the previous activation. Since that violates the definition of a busy window, the hypothesis must be rejected. □

The response time R_i of the q-th activation of a task in a busy window is given by:

$$R_i(q, S) = B_i(q, S) - \delta_i^-(q) \tag{6.20}$$

Finally, the worst-case response time R_i^+ is found inside the busy window and must be evaluated for all possible critical instant candidates $S \in \mathcal{S}$. The worst-case response time R_i^+ is given by:

$$R_i^+ = \max_{S \in \mathcal{S}} \left\{ \max_{1 \leq q \leq \eta_i^+(w_i(S))} \{R_i(q, S)\} \right\} \tag{6.21}$$

where the set $\mathcal{S}$ is given by the following Cartesian products:

$$\mathcal{S} = \left\{ \phi(\Gamma_j), \phi(\Gamma_k), \dots \right\} \times \left\{ \sigma_i(\Gamma_j) \times \sigma_i(\Gamma_k) \times \dots \right\} \tag{6.22}$$

where $\Gamma_j, \Gamma_k, \dots$ are all fork-join tasks mapped to the same core as τ_i and $\sigma_i(\Gamma_j)$ is the set of subtasks of Γ_j that are mapped to that core. When no fork-join tasks interfere with τ_i, $\mathcal{S} = \{(0, ())\}$.

Theorem 6.2. *R_i^+ (Equation 6.21) returns an upper bound on the worst-case response time of an independent task τ_i.*

Proof. We must first prove that, for a given S, R_i^+ is an upper bound. R_i^+ is obtained with the busy window approach [149]. It returns the maximum response time $R_i(q, S)$ among all activations inside the busy window. From Lemmas 6.7 and 6.8 we have that Equations 6.17 and 6.18 are upper bounds for a given S. From Lemma 6.9 we have that the busy window is captured by Equation 6.19. Since the first term of Equation 6.20 is an upper bound and the second term is a lower bound by definition, $R_i(q, S)$ is an upper bound. Thus R_i^+ is an upper bound for a given S. Since Equation 6.21 evaluates the maximum response time over all $S \in \mathcal{S}$, R_i^+ is an upper bound on the response time of τ_i. □

6.3.3. Error recovery

Designed for mixed-criticality, the approach supports different recovery strategies for different fork-join tasks (cf. Section 6.1). For instance, in DMR augmented with checkpointing and rollback, recovery consists in reverting the state and re-executing the error-affected stage in both replicas. In TMR, recovery consists in copying and replacing the state of the faulty

replica with the state of a healthy one. The different strategies are captured in the analysis by the recovery execution time, which depends on the strategy and the stage to be recovered. The recovery WCET $C_{i,rec}^{\sigma,s}$ of a fork-join subtask $\tau_i^{\sigma,s}$ accounts for the adopted recovery strategy as illustrated in Figure 6.3b. Once an error is detected, error recovery is triggered and executed in the recovery slot of the same cycle Φ. Figure 6.2 illustrates the recovery of the s-th stage of Γ_2's i-th activation.

The error recovery can now be integrated into the analysis. For a fork-join task Γ_i, the Multiple-Event Busy Time $B_i(q)$ (Equation 6.7) must be adapted to account for the execution of the recovery:

$$B_i^{rec}(q) = q \cdot s_i \cdot \Phi + j_\phi + \phi(recovery) - \phi(\Gamma_i) + C_{i,rec}^{\sigma,s} \tag{6.23}$$

where $C_{i,rec}^{\sigma,s}$ is the WCET of the recovery of last subtask of Γ_i. The recovery of another task Γ_j does not interfere with Γ_i's WCRT. Only the recovery of one of Γ_i's subtasks can interfere with Γ_i's WCRT. Moreover, since the recovery of a subtask occurs in the recovery slot of the same cycle Φ and does not interfere with the next subtask, only the recovery of the last stage of Γ_i actually has an impact on its response time. That is captured by the three last terms of Equation 6.23.

For an independent task τ_i, the worst-case impact of recovery of a fork-join task Γ_j is modelled as an additional fork-join task Γ_{rec} with one subtask $\tau_{rec}^{\sigma,1}$ mapped to the same core as τ_i and that executes in the *recovery* slot. The WCET $C_{rec}^{\sigma,1}$ of $\tau_{rec}^{\sigma,1}$ is chosen as the maximum recovery time among the subtasks of all fork-join tasks mapped to that core:

$$C_{rec}^{\sigma,1} = \max_{\forall \tau_j^{\sigma,s} \in hp_{FJ}(i)} \left\{ C_{i,rec}^{\sigma,s} \right\} \tag{6.24}$$

With Γ_{rec} mapped, Equation 6.21 finds the critical instant where the recovery $C_{rec}^{\sigma,1}$ has the worst impact on the response time of τ_i.

6.4. Experimental evaluation

The approach was experimentally evaluated with real as well as synthetic workload, focusing on the performance of the scheduler. First, MiBench benchmark applications [56] are characterized and evaluated as fork-join (replicated) tasks in the system. Then, the evaluation focuses on the performance of independent (regular) tasks. Finally, the approach is evaluated with synthetic workload when varying parameters of fork-join tasks.

6.4.1. Evaluation with benchmark applications

6.4.1.1. Characterization

First, the execution times and number of stages are extracted from MiBench automotive and security applications [56]. The applications were executed with small input on an ARMv7@1GHz and a DDR3-1600 [18]. Table 6.1 summarizes the total WCET, *observed* number of stages and WCET of the longest stage (max). A stage is delimited by *syscalls* (cf. Section 6.1). The observed execution times are reported as WCETs. As pointed out in [8], stages vary on number and execution time depending on the application and on the current activity in that stage (computation/IO). This is seen, e.g., in *susan*, where 99% of the WCET is concentrated in one stage (computation) while the other stages perform mostly IO and are on average 3.34us long.

Table 6.1.: MiBench applications' profile

	WCET [ms]	**Observed stages**		**Grouped stages**	
		#stages	max WCET [ms]	#stages	max WCET [ms]
basicmath	32.48	19738	0.02	5	6.50
bitcount	24.42	30	15.16	3	15.16
susan	9.63	12	9.59	1	9.63
blowfish	0.11	7	0.09	1	0.11
rijndael	13.17	93	0.37	3	5.91
sha	3.49	51	0.11	2	1.90

In the proposed approach, the optimal is when all stages of a fork-join task have the same WCET. There are two possibilities to achieve that: to *split* very long stages in smaller ones or to *group* small subsequent stages together. The latter is exploited here as it does not require changes to the error detection mechanism or to the proposed model. The result of grouping stages is that several stages are concurrently executed in a cycle Φ. Notice that it also includes several executions of state comparison since that occurs at the end of every stage.

The results with *grouped* stages are shown on the right-hand side of Table 6.1. The stages were first grouped without increasing the maximum stage length. The largest improvement is seen in *bitcount*, where the number of stages reduces in one order of magnitude. In cases where all stages

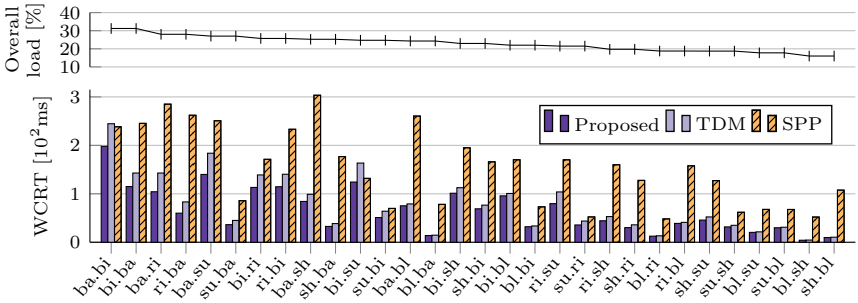


Figure 6.7.: WCRT of fork-join tasks with two segments derived from MiBench.

are very short, the maximum stage length was then increased. When increasing the maximum stage length in two orders of magnitude, the number of stages of *basicmath* reduces in four orders of magnitude. The maximum stage length was manually chosen. Alternatively, the problem of finding the maximum stage length can be formulated as an optimization problem that, e.g., minimizes the overall WCRT or maximizes the slack. Next, the benchmark applications are mapped as fork-join tasks and have their WCRTs evaluated.

6.4.1.2. Evaluation of fork-join tasks

Two applications at a time are mapped as fork-join tasks with two segments (i.e., replicas in DMR) to two cores (cf. Figure 6.4). On each core, 15% load is introduced by ten independent tasks generated with UUniFast [17]. The proposed approach is then compared with a TDM-based scheduler and Axer's Partitioned SPP [8]. In TDM, each fork-join task executes (and recovers) in its own slot. Independent tasks execute in a third slot, which replaces the recovery slot of the proposed approach. The size of the slots are derived from the offsets of the proposed approach. For all approaches, the priority assignment for independent tasks is deadline monotonic and considers that deadline equals period. In SPP, the deadline monotonic priority assignment also includes fork-join tasks.

The results are plotted in Figure 6.7, where *ba.bi* gives the WCRT of *basicmath* when mapped together with *bitcount*. Despite the low system

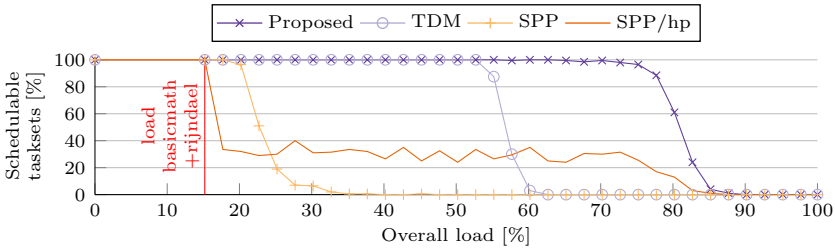
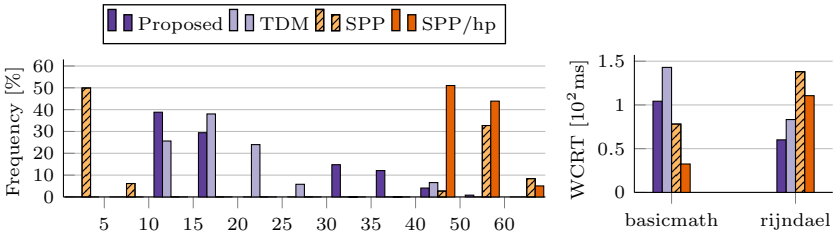


Figure 6.8.: Schedulability as a function of the load of the system. *Basicmath* and *rijndael* as fork-join tasks with two segments.

load, the proposed approach also outperforms SPP in all cases, with bounds 58.2% lower, on average. Better results with SPP cannot be obtained unless the interfering workload is removed or highest priority is given to the fork-join tasks [8], which violates DM. Despite the similarity of how the proposed approach handles fork-join tasks with TDM, it outperforms TDM in all cases, achieving, on average, bounds 13.9% lower. That minor difference is because TDM slots must be slightly longer than the offsets to fit an eventual recovery. Nonetheless, the proposed approach not only can guarantee short WCRT for replicated tasks but it also provides for the performance of independent tasks.

6.4.1.3. Evaluation of independent tasks

In a second experiment, *bitcount* and *rijndael* are fixed as fork-join tasks and load on both cores is varied. The generated task periods are in the range [20,500] ms, larger than the longest stage of the fork-join tasks. The schedulability of the system as the load increases is shown in Figure 6.8. The proposed approach outperforms TDM and SPP in all cases, scheduling 1.55x and 6.96x more tasksets, respectively. Due to its non-work conserving characteristic, TDM's schedulability is limited to medium loads. SPP provides very small response times with lower loads but, as the load increases, the schedulability drops fast due to high interference (and thus high WCRT) suffered by fork-join tasks. For reference purposes, the plot also shows the schedulability of SPP when assigning the highest priorities to the fork-join tasks (SPP/hp). The schedulability in higher loads improves but losing deadline monotonicity guarantees renders the systems



(a) WCRT of independent tasks [ms] (b) WCRT of FJ tasks

Figure 6.9.: *Basicmath* and *rijndael* as replicated tasks in DMR running on a dual-core configuration with 20.2% load (5% load from independent tasks).

unusable in practice. Moreover, when increasing the jitter to 20% (relative to period), schedulability decreases 14.2% but shows the same trends for all schedulers.

Figure 6.9 details the tasks' WCRTs when the system load is 20.2%. Indeed, when schedulable, SPP provides some of the smallest WCRTs for independent tasks, and SPP/hp improves the response times of fork-join tasks at the expense of the independent tasks'. The proposed approach provides a balanced trade-off between the performance of independent tasks and of fork-join tasks, and achieves high schedulability even in higher loads.

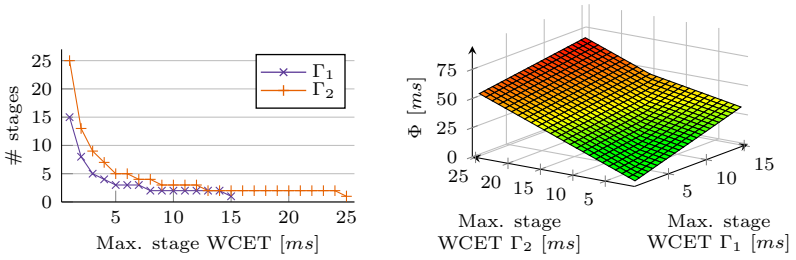
6.4.2. Evaluation with synthetic workload

The performance of the proposed approach was also evaluated when varying parameters such as stage length and cycle Φ.

6.4.2.1. Evaluation of fork-join tasks

Two fork-join tasks Γ_1 and Γ_2 with two segments each (i.e., replicas in DMR) are mapped to two cores. The total WCETs[3] of Γ_1 and Γ_2 are 15 and 25ms, respectively. Both tasks are sporadic, with a minimum distance of 1s between activations. The number of stages of Γ_1 and Γ_2 is varied as a function of the maximum stage WCET, as depicted in Figure 6.10a. The

[3]The sum of the WCET of all stages of a fork-join task.



(a) Stages of Γ_1 and Γ_2 (b) Cycle Φ

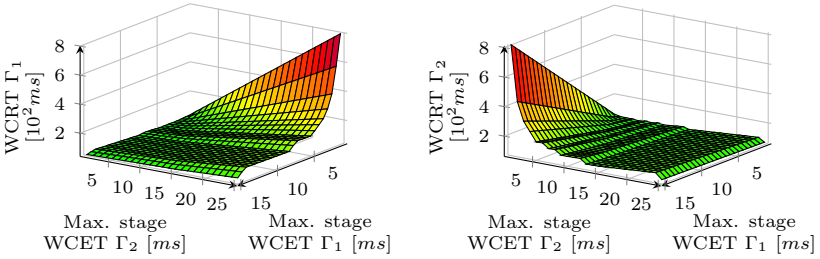
Figure 6.10.: Parameters of two fork-join tasks Γ_1 and Γ_2 with two segments running on a dual-core configuration.

length of the cycle Φ, depicted in Figure 6.10b, varies with the maximum stage WCET since it is derived from them (cf. Section 6.2.4).

The system performance as the maximum stage lengths of Γ_1 and Γ_2 increase is reported in Figure 6.11. The WCRT of Γ_1 increases with the stage length (Figure 6.11a) as it depends on the number of stages and Φ's length. In fact, the WCRT of Γ_1 is longest when the stages of Γ_1 are the shortest and the stages of the interfering fork-join task (Γ_2) are the longest. Conversely, WCRT of Γ_1 is shortest when its stages are the longest and the stages of the interfering fork-join task are the shortest. The same occurs to Γ_2 in Figure 6.11b. Thus, there is a trade-off between the response times of interfering fork-join tasks. That is plotted in Figure 6.12 as the sum of the WCRTs of Γ_1 and Γ_2. As can be seen in Figure 6.12, low response times can be obtained next and above to the line segment between the origin $(0, 0, 0)$ and the point $(15, 25, 0)$, the total WCETs[1] of Γ_1 and Γ_2 respectively.

6.4.2.2. Evaluation of independent tasks

To evaluate the impact of the parameters on independent tasks, the previous scenario was extended introducing 25% load on each core with ten independent tasks generated with UUniFast [17]. The task periods are within the interval [15,500] ms for the first experiment, and the interval [25,500] ms for the second. The priority assignment is deadline monotonic and considers that the deadline is equal to the period.



(a) WCRT of Γ_1 (b) WCRT of Γ_2

Figure 6.11.: Performance of fork-join tasks Γ_1 and Γ_2 as a function of the maximum stage WCET.

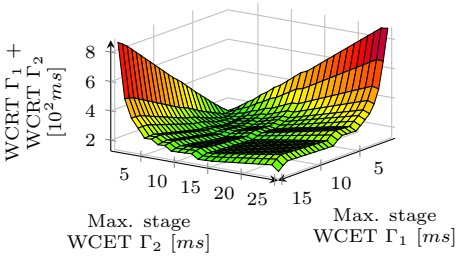
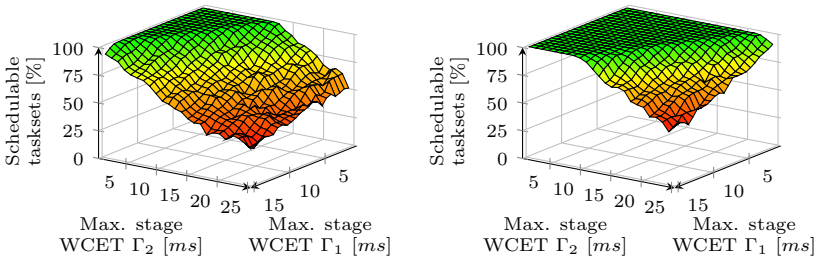


Figure 6.12.: WCRT trade-off between interfering fork-join tasks.

The schedulability as a function of the stage lengths is shown in Figure 6.13. Sufficiently long stages cause the schedulability to decrease as independent tasks with short periods start missing their deadlines. That is seen in Figure 6.13a when the stage length of either fork-join task reaches 15ms, the minimum period for the generated tasksets. Thus, when increasing the minimum period of generated tasks to 25ms, the number of schedulable tasksets also increases (Figure 6.13b).

The maximum stage length of a fork-join task has direct impact on the response times and schedulability of the system. For the sake of performance, shorter stage lengths are preferred. However, that is not always possible because it would result in a large number of stages or because of the application, which restricts the minimum stage length (cf. Section 6.4.1.1). Nonetheless, fork-join tasks still are able to perform well with appropriate



(a) Task period interval [15-500]ms

(b) Task period interval [25-500]ms

Figure 6.13.: Schedulable tasksets as a function of the maximum stage WCET of fork-join tasks Γ_1 and Γ_2 with 25% load from independent tasks.

parameter choices. Additionally, one can formulate the problem of finding the stage lengths according to an objective function, such as minimize the overall response time or maximize the slack. The offsets can also be included in the formulation, as long as Constraints 6.1 and 6.2 are met.

6.5. Cross-layer integration discussion

Both the resilient NoC and the AIQ mechanism, introduced in Chapters 3 to 5, can be combined and used with replicated execution and the replica-aware co-scheduling in a cross-layer approach. However, their application depend on the system requirements, as discussed next.

When error recovery is a requirement, the replicated execution approach requires a reliable on-Chip communication, as described in Section 6.2.3. That is due to the fact that the approach relies on critical components such as the voters and the RTOS kernel – also referred to as the reliable computing base – which are assumed to be reliable [45, 126]. The resilient NoC can be employed to provide reliable on-Chip communication and protect therewith the critical communication – e.g., voter communication.

The NoC traffic of protected and unprotected tasks can be protected with either the resilient NoC and AIQ. Since tasks are protected by the replicated execution approach and error recovery is therefore performed

in software, hardware error detection suffices. Also, unprotected tasks do not require protected communication. Thus, the choice between both approaches can be guided by the target reliability and timing requirements in a "good enough" strategy, as discussed in Section 5.4.4. N.B. in case of NoC resets with the AIQ approach, a controlled NoC reset must be employed to prevent the corruption of unaffected NoC transactions, which would wrongly induce multiple error scenarios requiring the recovery of potentially all replicated tasks.

Another use of replicated execution is for providing integrity only, in a *integrity* mode. In that mode, replicas can be used for error detection only, without implementing any recovery strategy. When an error is detected – i.e., replicas differ – the error is reported to the next layer in the hierarchical fault-tolerant architecture (cf. Section 1.5.2). Notice that, providing only integrity facilitates error in that SDCs will eventually be detected when they manifest themselves, but the system must not recover from them. Consider, for instance, the problem of SDCs and checkpointing: one must ensure the absence of SDCs in a checkpoint's state, otherwise recovery will fail.

Another possibility is the combination of both approaches, ARQ and AIQ, in the same NoC. Thus, the NoC traffic of critical components, such as the voters, can rely on error recovery in hardware the reliable NoC communication. Replicated tasks, whose recovery is already guaranteed by the approach in software, can rely only on hardware error detection. The synergy in the combination of both approaches, in terms of hardware implementation requirements, has been envisioned as promising but has not been pursued in this thesis. The synergy can occur in that the AIQ approach with reset capabilities can be coupled with ARQ protocols, with retransmission buffers, for NoC traffic with different requirements. The former is used to guarantee traffic integrity with error detection in hardware, whereas the latter is used to guarantee traffic integrity and reliable service with error detection and recovery in hardware. Although the different types of services is already provided in the resilient NoC approach, the envisioned solution has potential to reduce the hardware implementation overhead without the use of reliable routers, as in the AIQ approach with reset capabilities. Further investigation in terms of hardware implementation is left as future work.

6.6. Summary

This chapter introduced the replica-aware co-scheduling for mixed-criticality, developed in the context of the project ASTEROID. The approach targets mixed-critical systems where applications with different requirements and criticalities co-exist in a cross-layer fault-tolerance approach. Upon the occurrence of errors due to random hardware faults, the error effects are contained and the appropriate error handling strategy is applied in software – e.g., recovery by rolling back or by restarting a task. Errors caused by random hardware faults are detected via hardware assisted error detection, such as pipeline fingerprinting and the AIQ mechanism, introduced in Chapter 5.

The proposed co-scheduling approach provides for high worst-case performance of replicated software execution on many-core architectures without jeopardizing the performance of the remaining tasks in the system. A formal WCRT analysis was presented, which supports different recovery strategies and accounts for the NoC communication delay and overheads due to replica management and state comparison. Experimental results with benchmark applications showed an improvement on taskset schedulability of up to 6.9x when compared to Partitioned SPP and 1.5x when compared to a TDM-based scheduler, both widely employed in the real-time domain.

7. Conclusion

Dependability has many facets to be addressed when designing a system. That is specially challenging in real-time mixed-critical systems, where safety standards play an important role and where responding in time can be as important as responding correctly or even responding at all. Dependability is threatened by random hardware faults, which have increased in importance to the point that they must be explicitly addressed. The key to efficiently address those challenges and achieve dependability are cross-layer approaches. This thesis has addressed the issue of cross-layer fault-tolerance approaches for achieving dependability of real-time mixed-critical systems. Three important requirements of such systems directed this work: *integrity*, *resilience* and *real-time*.

Chapter 2 described the impacts of random hardware faults causing soft errors on a real-time many-core system, from the NoC-level up to the system-level. Unhandled errors in the NoC can cause not only data corruption, packet loss and derouting, as usually considered in the literature, but it can also present static effects that lead to continuous corruption and blocking scenarios during runtime. Moreover, errors can propagate and indirectly affect the background traffic, propagating between criticalities and throughout the system. Such errors, when not appropriately handled, hinders the system's integrity and will likely overwhelm the system's error recovery capability. At the system-level, the impact of soft errors can range from masked, to wrong output, up to a complete system crash. Timing wise, errors can cause the output of an application to delay, independently of the output correctness, to the point where it becomes irresponsive. Based on those findings, a set of mechanisms to attain dependability were then introduced.

A resilient NoC was proposed in Chapter 3, which is able to transparently provide soft error protection to software with high reliability. The approach combines well known techniques into a strong fault containment with a resilient router design to limit the impact of errors in time and in space. When employed together with ARQ-based protocols, the resilient

NoC tolerates soft errors and provides reliable service with integrity and real-time guarantees. ARQ-based protocols and their efficient use in NoCs under real-time guarantees was addressed in Chapter 4. The presented transport layer analysis of protocols in NoCs was integrated with network layer analyses in such a way that it can be reused with different models of NoC with different arbitration policies. Moreover, it was shown that for cache-based memory traffic, which present low burstiness, simple ARQ protocols perform well. For bursty, throughput intensive traffic, such as in DMA transfers, it was shown that optimized protocols, such as the proposed DMA ARQ protocol, can deliver superior performance while being independent of expensive retransmission buffers.

The thesis also explored a low-overhead alternative for NoCs to guarantee integrity and timing under errors. Chapter 5 introduced AIQ, an end-to-end mechanism to detect errors in NoCs. AIQ explores the idea of keeping track of transactions of distributed systems in the context of NoCs for predictable real-time systems. AIQ integrates into the NoC resulting in a low-overhead fault-tolerant NoC capable of detecting errors and ensuring that their effects are contained in time. Error handling is delegated to the higher layers of hardware and software, which may react according to an arbitrary strategy in a cross-layer approach.

Finally, Chapter 6 addressed the problem of efficiently handling errors in software with a replica-aware co-scheduling approach for mixed-criticality. The approach targets cross-layer fault-tolerant systems providing reliable software execution on unreliable hardware by means of redundant software execution. Errors are detected via hardware assisted error detection and error handling is performed in software. The error handling strategies can vary from simple fault-containment to roll-back and re-execute and state copy depending on the system configuration. The proposed replica-aware co-scheduling provides for the high performance of replicated software execution in many-core platforms without jeopardizing the performance of the remaining tasks. Unlike conventional real-time schedulers, which are severely affected task synchronization overheads, co-scheduling has shown to achieve high resource utilization and enables the use of replicated execution in practice.

Future real-time systems are migrating to many-core platforms looking for more efficiency, computational power and extended lifetime. The quest for providing performance guarantees for different functions and criticality levels within those platforms without wasting resources is still open with

no consensus. With respect to dependability, there are many open challenges and, with them, opportunities. One clear path is that the abundant hardware available in many-cores can be exploited for increased reliability in cross-layer approaches. The key element that enables one to efficiently achieve dependability is *integrity* and *error detection*. Error detection will definitely require hardware support for efficiency and low impact on performance. Error handling will most likely be handled in software, where error handling and recovery strategies can be flexible and easily adaptable. Extensions with respect to hard errors, might lead to reconfigurable hardware in the direction of bio-inspired, self-healing systems, requiring therefor hardware architectures with higher degrees of configurability. Those systems can be not only reactive but also proactive in that the system's operation can be guided so as to extend the system's lifetime. The future challenge in such highly adaptable mixed-critical systems is then to ensure the guarantees of critical tasks at all times as the system ages and new configurations, which cannot be defined at design time, are required.

The envisioned future work includes the synergy in the combination of the developed approaches. As seen in the redundant software execution approach, replicated tasks are protected by the approach, but there is always the assumption that certain critical hardware and software components perform reliably. The synergy can occur in that the AIQ approach with reset capabilities can be coupled with ARQ protocols, with retransmission buffers, for NoC traffic with different requirements. The former is used to guarantee traffic integrity with error detection in hardware, whereas the latter is used to guarantee traffic integrity and reliable service with error detection and recovery in hardware. Although the different types of services is already provided in the resilient NoC approach, the envisioned solution has potential to reduce the hardware implementation overhead without the use of reliable routers. Thus, a combination between error detection and recovery in the same NoC with ARQ and AIQ can be explored in terms of hardware and software implementation.

A. Protocol Definitions

This appendix presents the finite state machines of the protocols addressed in Chapter 4. It also reproduces relevant parts of the formal analysis of Go-Back-N ARQ of Axer.

ARQ-based protocols when employed above the data-link layer are connection oriented. Each connection requires a pair of protocols instances: one at the sender and one at the receiver. Notice that multiple connections between a same sender-receiver pair are also possible. Thus, throughout this appendix, the term *sender* will be used to refer to the respective protocol instance at the sender. Similarly, the term *receiver* will be used to refer to the respective protocol instance at the receiver. Notice that connection and traffic stream can be used interchangeably with the concept used throughout this Thesis.

In order to identify that a packet belongs to a specific connection a triple $\{senderID, receiverID, connectionID\}$ is employed, where the first two are, e.g., MAC addresses of the sender and receiver, respectively, and the last is the ID of that particular connection. The connectionID can be dropped when only single connection per sender-receiver pair are allowed.

To identify different packets within the same connection, sequence numbers are employed. Those can be seen, e.g., in the example of Figure 4.4. Sequence numbers are usually managed in a circular fashion and the length of the sequence depends on the protocol. For instance, Stop-and-Wait can operate with only two sequence numbers, as there will be at most two different packets in-flight at any given time during operation (not shown in Figure 4.4 in favor of demonstrating the progress of the transmission). Go-Back-N, on the other hand, requires a longer sequence, since there might exist in-flight in the network.

A.1. Stop-and-Wait ARQ

This section presents the finite state machine diagram of Stop-and-Wait ARQ considered in this work and whose worst-case behavior is captured by the analysis of Axer.

Figure A.1 presents the state machine of the sender. When the sender is *idle* and there is a packet to be transmitted (*has_pkt*), it is immediately transmitted (*transmit*) and the sender starts waiting for an ACK. In the *wait 4 ACK* state, a timer is started. If an ACK is received before the time reaches a timeout, the sender either goes back to *idle* or transmits the next packet (*transmit*) depending on whether there is a packet ready to be transmitted (*has_pkt*). In case of a timeout, a retransmission of the last transmitted packet is performed (*retrans.*) and the sender resumes waiting for an ACK (*wait 4 ACK*).

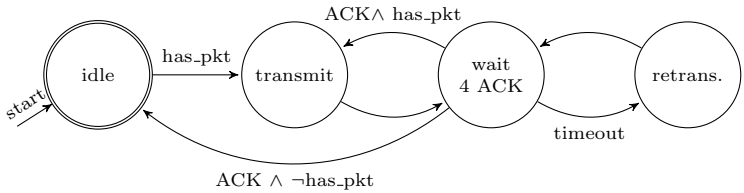


Figure A.1.: The Stop-and-Wait ARQ state machine: sender.

Monitoring can be introduced in the *wait 4 ACK* state to monitor the current conditions of the network, including RTT and error rates. With that, impending timing violations can be detected and reported to the system while ensuring integrity.

Whenever the protocol instance is not *idle* and a new packet to be transmitted arrives, it is assumed to be stored in a buffer, forming a backlog. The buffer also enables *has_pkt* when it is not empty. Alternatively, in practice, specially in NoCs, where strong constraints on circuit area and power consumption dominate, instead of storing new packets in a buffer, the protocol instance can stall new packets until it becomes *idle* again. In terms of worst-case response time, there will be no difference as long as tasks wait for the completion of the transmission instead of yielding after pumping packets into the buffer but before they can actually be pumped into the network.

The state machine of Stop-and-Wait on the receiver side is presented in Figure A.2. When *idle* and a packet arrives (*has_pkt*), the receiver checks its sequence number (*check*). When the packet is the *next* packet in the sequence, it is received and acknowledged. Otherwise, if the packet is an older one, it is dropped but acknowledged nonetheless. That is required, for instance, due to cases where the ACK is lost and a same packet is successfully received twice. Notice that a state does not necessarily translate into a cycle of the pipelined implementation in hardware – e.g., receiving and acknowledging the received packet could be performed in parallel.

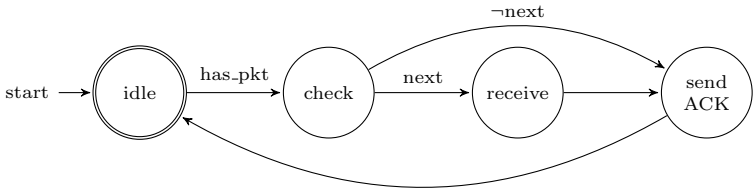


Figure A.2.: The Stop-and-Wait ARQ state machine: receiver.

A.2. Go-Back-N ARQ

This Section presents the finite state machine diagram of Go-Back-N ARQ considered in this work and whose worst-case behavior is captured by the analysis of Axer.

Figure A.3 presents the state machine of the sender. When the sender is *idle* and there is a packet to be transmitted (*has_pkt*), the sender immediately transmits it (*transmit*) and continues transmitting packets as long as there are packets to be sent (*has_pkt*) and the send window has not been fully utilized (¬*window_full*). Whenever a packet is transmitted, a slot in the send window is allocated for it and the respective timer is started. If the send window has been fully utilized (*window_full*) or there are no more packets to transmit (¬*has_pkt*), the sender starts waiting for ACKs (*wait 4 ACK*). Notice that ACKs can be received any time when the sender is in one the states: *transmit*, *wait 4 ACK* or *retrans. window*. The receipt of an ACK releases the respective slot in the send window and stops its timer. In the *wait 4 ACK* state, the sender will either start transmitting

again if there is backlog and if the window is not full anymore, or it will return to idle after all transmitted packets have been acknowledged. In case a timeout expires before the respective ACK is received, the sender starts retransmitting the entire send window (*retrans. window*), starting with the oldest unacknowledged packet. When the retransmission is *done*, the sender returns to the *wait 4 ACK* state.

Similarly to Stop-and-Wait ARQ, monitoring can be introduced in the *wait 4 ACK* state to monitor the current conditions of the network, including RTT and error rates. With that, impending timing violations can be detected and reported to the system while ensuring integrity.

Backlog is handled as described above in Stop-and-Wait ARQ.

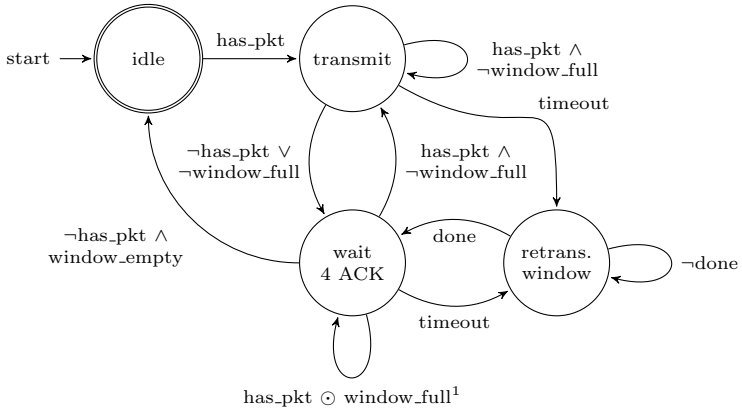


Figure A.3.: The Go-Back-N ARQ state machine: sender.

The state machine of the receiver is presented in Figure A.4. It is similar to Stop-and-Wait, it differs in the logic in checking and acknowledging. In Go-Back-N, the most simple implementation of *check* while providing in-order delivery, as shown in Algorithm A.1, is to receive only the next packet in the sequence. Packets that are not the next in the sequence are dropped and not acknowledged. Packets that have already been received (older packets in the sequence) are dropped, since they were already received, and acknowledged.

Notice that there are possible improvements to the protocol that are able

[1]The binary operator $\odot$ is the *negated exclusive OR*. Thus $A \odot B \Leftrightarrow (A \wedge B) \vee (\neg A \wedge \neg B)$.

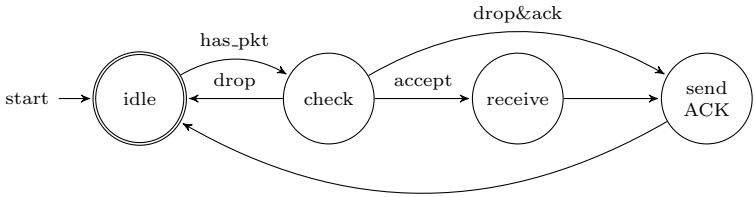


Figure A.4.: The Go-Back-N ARQ state machine: receiver.

Algorithm A.1 Go-Back-N simple check

INPUT: incoming packet p belonging to connection c
OUTPUT: $x \in \{accept, drop\&ack, drop\}$

```
1: let seqNum(p) return the sequence number of packet p
2: let nSeqNum(c) return the next sequence number of connection c
3: if seqNum(p) = nSeqNum(c) then
4:     return accept // Accept with acknowledgement
5: else if seqNum(p) < nSeqNum(c) then
6:     return drop&ack // Drop with acknowledgement
7: else
8:     return drop // Drop without acknowledgement
9: end if
```

to improve the average case. However, the wost-case remains the same. For that reason such improvements were not further investigated. One possible improvement is to not retransmit packets that were already acknowledged. That requires the receiver to accept and acknowledge packets even when a previous packet is missing. Packets can be either delivered potentially out-of-order to the application or delivered in-order with the help of a reorder buffer at the receiver.

The next sections reproduce relevant parts of the formal analysis of Go-Back-N ARQ of Axer. The analysis is divided in two parts: the error-free case and the error case. Proofs are skipped and can be found in [8].

A.2.1. Formal analysis: the error-free case

The analysis relies on the busy window approach [149]. It starts by deriving the *busy period* for the protocol.

The busy period $w_{gbn,i}$ is the largest time interval in which packets arrive at the transport layer and need to be queued while the ARQ protocol is waiting for ACKs of previous packets. It is given by:

$$w_{gbn,i} = \left\lceil \frac{\eta^+_{tx,i}(w_{gbn,i})}{n_{sw}} \right\rceil \cdot RTT^+_i \tag{A.1}$$

Equation A.1 forms an integer fixed point problem typical in busy-window-based analyses – $w_{gbn,i}$ is present on both sides of the equation. The problem can be solved iteratively, starting with a very small $\epsilon > 0$ ($w_{gbn,i} = \epsilon$).

The Worst-Case Multiple Packet Forwarding Time $B^+_{gbn,i}(q)$ is the largest time interval to forward a sequence of q packets, assuming error-free conditions and considering that all but the first packet arrive before the previous packet is acknowledged. $B^+_{gbn,i}(q)$ of a stream i is given by:

$$B^+_{gbn,i}(q) = (q-1) \cdot RTT^+_i \tag{A.2}$$

The Best-Case Multiple Packet Forwarding Time $B^-_{gbn,i}(q)$ is the smallest time interval to forward any sequence of q packets. It is given by:

$$B^-_{gbn,i}(q) = \left(\left\lceil \frac{q}{n_{sw}} \right\rceil - 1 \right) \cdot RTT^-_i \tag{A.3}$$

The Worst-Case Response Time $R^+_{gbn,i}$ is the largest time interval that any packet is delayed by Go-Back-N ARQ before being forwarded to the network. $R^+_{gbn,i}$ of a stream i is bounded by:

$$R^+_{gbn,i} = \max_{1 \le q \le \tilde{\eta}^+_{tx,i}(w_{gbn,i})} \{B^+_{gbn,i}(q) - \delta^-_{tx,i}((q-1) \cdot n_{sw} + 1)\} \tag{A.4}$$

with

$$\tilde{\eta}^+_{tx,i}(\Delta t) = \left\lceil \frac{\eta^+_{tx,i}(\Delta t)}{n_{sw}} \right\rceil \tag{A.5}$$

where $\tilde{\eta}^+_{tx,i}(\Delta t)$ is an upper bound on the number of packet that belong to the same group.

The output event model capturing the traffic injection in the network by Go-Back-N ARQ $\delta^-_{gbn,tx}$ under error-free conditions is obtained as follows:

$$\delta^-_{gbn,tx}(q) = \max\{\delta^-_{tx}(q) - R^+_{gbn} + R^-_{gbn}, B^-_{gbn}(q-1)\} \tag{A.6}$$

The *overall latency* $\mathbb{L}^{+}_{gbn,i}(q)$ for transferring q data packets in a stream i is given by:

$$\mathbb{L}^{+}_{gbn,i}(q) = \delta^{-}_{tx,i}(q) + L^{+}_{i} + R^{+}_{gbn,i} \tag{A.7}$$

A.2.2. Formal analysis: the error case

Similarly to the analysis in Section 4.3.3.2, the worst-case impact of an error in Go-Back-N is when it causes the loss of an ACK just before it is delivered back to the sender.

First, the impact of errors is integrated into the busy period. The k-error busy period $w_{gbn,i}(k)$ is given by:

$$w_{gbn,i}(k) = k \cdot (t_{out} + RTT^{+}_{i}) + \left\lceil \frac{\eta^{+}_{tx,i}(w_{gbn,i}(k))}{n_{sw}} \right\rceil \cdot RTT^{+}_{i} \tag{A.8}$$

In the worst case, in addition to the error-free busy window, each error leads to a timeout t_{out} and a retransmission RTT^{+}_{i} (data packet and ACK).

Similarly, the multiple packet forwarding time under errors is derived accounting for the impact of k errors on the transfer. The k-error Worst-Case Multiple Packet Forwarding Time $B^{+}_{gbn,i}(q,k)$ is given by:

$$B^{+}_{gbn,i}(q,k) = k \cdot (t_{out} + RTT^{+}_{i}) + (q-1) \cdot RTT^{+}_{i} \tag{A.9}$$

In addition to $B^{+}_{gbn,i}$, in the worst case, each error leads to a timeout and a retransmission whose impact is also bounded by $t_{out} + RTT^{+}_{i}$, as in Equation A.8.

The k-error Worst-Case Response Time $R^{+}_{gbn,i}(k)$ is derived from Eq. A.4 to account for the k-error versions of $w_{gbn,i}(k)$ and $B^{+}_{gbn,i}(q,k)$:

$$R^{+}_{gbn,i}(k) = \max_{1 \le q \le \tilde{\eta}^{+}_{tx,i}(w_{gbn,i}(k))} \{B^{+}_{gbn,i}(q,k) - \delta^{-}_{tx,i}((q-1) \cdot n_{sw} + 1)\} \tag{A.10}$$

where $\tilde{\eta}^{+}_{tx,i}(\Delta t)$ is given by Equation A.5.

The *k-error overall latency* $\mathbb{L}^{+}_{gbn,i}(q,k)$ for transferring q data packets in a stream i is given by:

$$\mathbb{L}^{+}_{gbn,i}(q,k) = \delta^{-}_{tx,i}(q) + L^{+}_{i} + R^{+}_{gbn,i}(k) \tag{A.11}$$

Besides the latency in the error-free case (Equation 4.15), in the k-error scenario, Equation 4.19 must account for the latency overhead. This is included by the last term, the k-error worst-case response time $R^{+}_{dma,i}(k)$.

A.3. DMA ARQ

This Section presents the finite state machine diagram of DMA ARQ, which was introduced and formally analyzed in Section 4.3.3.

Figure A.5 presents the state machine of the sender. When the sender is *idle* and there is a new *transfer*, the sender immediately starts transmitting (*transmit*) and continues transmitting packets as long as there are packets to be sent (*has_pkt*). When there are no more packets to transmit ($\neg$*has_pkt*), the sender starts waiting for ACKs from the receiver (*wait 4 ACK*). In the *wait 4 ACK* state, the sender will either return to idle after receiving and ACK or start a retransmission if a *timeout* occurs or if a NACK is received. In a retransmission (*retrans.*), the data is fetched from the memory for as many packets as required (*access memory*). The content that is retransmitted depends on event triggering it: in case of a *timeout*, the last transmitted packet is retransmitted; in case of a NACK, the missing data requested by it. When the retransmission is *done*, the sender returns to the *wait 4 ACK* state.

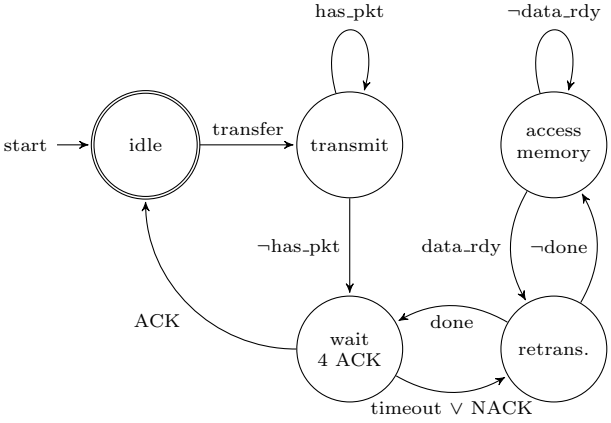


Figure A.5.: The DMA ARQ state machine: sender.

Similarly to Stop-and-Wait and Go-Back-N ARQ, monitoring can be introduced in the *wait 4 ACK* state to monitor the current conditions of the network, including RTT and error rates. With that, impending timing violations can be detected and reported to the system while ensuring

integrity.

The state machine of the receiver is presented in Figure A.6. When *idle* and a new transfer starts, the receiver transitions to *receive* and remains there until all packets of the transfer are received. When the last packet of the transfer is received (*last_pkt*), the receiver sends either an ACK or a NACK depending on whether all packets were successfully received or not, respectively. In case one or more packets were missing, the NACK requests the retransmission of those packets and the receiver transitions to *receive retrans.* and remains there until all missing packets are received. Notice that, when the last packet of a transfer is lost, the receiver remains in the *receive* state until receiving the packet retransmitted by the sender. The same applies for the last packet of a retransmission in the *receive retrans.* state.

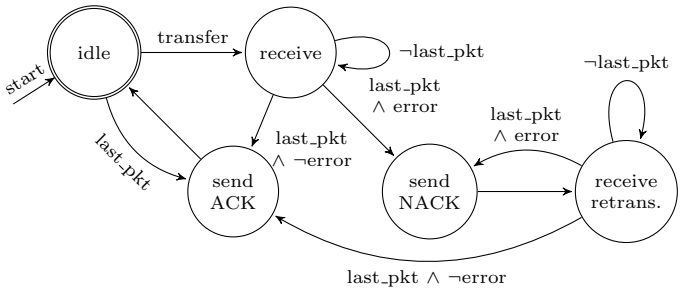


Figure A.6.: The DMA ARQ state machine: receiver.

Notice that the transition loop in the *receive* and *receive trans.* states can lead to a deadlock in case of a network failure – i.e., the transfer does not finish and prevents other transfers to take place. Although a second transfer in presence a network failure would most likely fail, the detection of the network conditions is relevant. Just as in any ARQ-based protocol, the network conditions can and should be monitored in order to detect unrecoverable errors, such as permanent faults or transient faults with static effects, and to detect error scenarios that lead to a timing violation, such as in the case of error rates higher than expected.

B. List of Publications

This appendix lists all publications by the author. The publication are divided into those related to the thesis and those unrelated. They are listed in inverse chronological order of publication.

B.1. Publications related to this thesis

B.1.1. Reviewed

E. A. Rambo, C. Seitz, S. Saidi, and R. Ernst. Bridging the Gap between Resilient Networks-on-Chip and Real-Time Systems. In *IEEE Transactions on Emerging Topics in Computing (TETC)*, 2017 [131]. This article is an extended version of [132]. The major extensions are with respect to the reliable transport with ARQ-based protocols and in the experimental evaluation. It features a more comprehensive reliability evaluation, a comparison with a TMR approach and an additional avionics use case. Its contents are featured in Chapter 3.

E. A. Rambo and R. Ernst. Replica-Aware Co-Scheduling for Mixed-Criticality. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, volume 76, pages 20:1–20:20, 2017 [127]. This article addressed the cross-layer approach with replicated software execution and proposed the replica-aware co-scheduling for mixed-criticality for its performance improvement. The approach is introduced in Chapter 6.

E. A. Rambo, C. Seitz, S. Saidi, and R. Ernst. Designing networks-on-chip for high assurance real-time systems. In *IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC'17)*, 2017 [132]. This article addressed the issue of a NoC for high assurance real-time systems and proposed the resilient NoC that is introduced in Chapter 3.

E. A. Rambo, S. Saidi, and R. Ernst. Providing formal latency guarantees for ARQ-based protocols in Networks-on-Chip. In *Design, Automation*

& Test in Europe Conference & Exhibition (DATE'16), 2016 [130]. This article addressed the use of ARQ-based protocols in NoCs. It presented a transport layer worst-case communication time analysis which can be integrated with network layer analyses of NoCs, including packet-switched and wormhole-switched ones. Its contents are featured in Chapter 4.

E. A. Rambo and R. Ernst. Providing flexible and reliable on-chip network communication with real-time constraints. In *1st International Workshop on Resiliency in Embedded Electronic Systems (REES)*, 2015 [126]. This article introduced the idea of a resilient and highly reliable NoC, which was then realized in [130, 132, 131].

E. A. Rambo and R. Ernst. Worst-case communication time analysis of networks-on-chip with shared virtual channels. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'15)*, 2015 [125]. This article addressed the worst-case communication time analysis problem of NoC at the network layer and proposed an extension to [33] by supporting shared VCs and reducing conservativeness. The analysis proposed in this article was used as basis for the integration with the transport layer analysis and the respective experiments in Chapter 4.

E. A. Rambo, A. Tschiene, J. Diemer, L. Ahrendts, and R. Ernst. FMEA-Based Analysis of a Network-on-Chip for Mixed-Critical Systems. In *Eighth IEEE/ACM International Symposium on Networks-on-Chip (NOCS'14)*, 2014 [129]. This article is an extended version of [128]. The major extensions are an assessment considering the occurrence probability of failure modes and an improvement to the router architecture which results in reduced cases of static effects. Its contents are featured in Chapter 2. The outcome and insight provided by this work is used as input for the techniques developed throughout the thesis,

E. A. Rambo, A. Tschiene, J. Diemer, L. Ahrendts, and R. Ernst. Failure analysis of a Network-on-Chip for real-time mixed-critical systems. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'14)*, 2014 [128]. This article addressed the impacts of soft errors on NoCs and proposed the FMEA-based analysis that is compliant with safety standards. Its contents are featured in Chapter 2.

B.1.2. Under review

E. A. Rambo, Y. Shang, and R. Ernst. Providing Integrity in Real-Time Networks-on-Chip. Submitted to *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 2018 [133]. This article introduces the AIQ mechanism for ensuring integrity and real-time guarantees with low overhead in real-time NoCs. Its contents are featured in Chapter 5.

B.1.3. Unreviewed

E. A. Rambo, L. Ahrendts, and J. Diemer. FMEA of the IDAMC NoC. Technical report, Institute of Computer and Network Engineering – TU Braunschweig, 2013 [124]. This technical report contains the verbose output of the analysis proposed and reported in Chapter 2 and in [128, 129].

B.2. Publications not related to this thesis

T. Kadeed, E. A. Rambo, and R. Ernst. Power and area evaluation of a fault-tolerant Network-on-Chip. In *30th IEEE International System-on-Chip Conference (SOCC'17)*, 2017 [74].

T. Stumpf, H. Härtig, E. A. Rambo, and R. Ernst. Cross-layer Resilience Mechanisms to Protect the Communication Path in Embedded Systems. In *1st International Workshop on Resiliency in Embedded Electronic Systems (REES)*, 2015 [144].

L. S. Freitas, E. A. Rambo, and L. C. V. dos Santos. On-the-fly verification of memory consistency with concurrent relaxed scoreboards. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'13)*, 2013 [51].

E. A. Rambo, O. P. Henschel, and L. C. V. dos Santos. On ESL verification of memory consistency for system-on-chip multiprocessing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'12)*, 2012 [123].

E. A. Rambo, O. P. Henschel, and L. C. V. dos Santos. Automatic generation of memory consistency tests for chip multiprocessing. In *18th IEEE International Conference on Electronics, Circuits, and Systems (ICECS'11)*, 2011 [122].

List of Figures

List of Tables

List of Algorithms

List of Acronyms

Bibliography

[1] Leonie Ahrendts. Design und implementierung von fehlertoleranzmechanismen für ein Network-on-Chip. Bachelor's thesis, TU Braunschweig, March 2012.

[2] Konstantinos Aisopos, Chia-Hsin Owen Chen, and Li-Shiuan Peh. Enabling system-level modeling of variation-induced faults in networks-on-chips. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 930–935, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0636-2. doi: 10.1145/2024724.2024931.

[3] Björn Andersson and Dionisio de Niz. Analyzing Global-EDF for multiprocessor scheduling of parallel tasks. In *International Conference On Principles Of Distributed Systems*, pages 16–30. Springer, 2012.

[4] Arteris. FlexNoC® Interconnect IP. `http://www.arteris.com/flexnoc`, 2018. [Online; accessed 04-May-2018].

[5] Arteris. Ncore™ Cache Coherent Interconnect IP. `http://www.arteris.com/ncore`, 2018. [Online; accessed 04-May-2018].

[6] JL Autran, P Roche, S Sauze, G Gasiot, D Munteanu, P Loaiza, M Zampaolo, and J Borel. Altitude and underground real-time SER characterization of CMOS 65nm SRAM. *IEEE Transactions on Nuclear Science*, 56(4), 2009.

[7] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[8] Philip Axer. *Performance of Time-Critical Embedded Systems under the Influence of Errors and Error Handling Protocols*. PhD thesis, TU Braunschweig, 2015.

[9] Philip Axer, Maurice Sebastian, and Rolf Ernst. Reliability analysis for mpsocs with mixed-critical, hard real-time constraints. In *Proc. Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011.

[10] Philip Axer, Rolf Ernst, Björn Döbel, and Hermann Härtig. Designing an analyzable and resilient embedded operating system. In *Proc. on Software-Based Methods for Robust Embedded Systems*, Braunschweig, Germany, 2012.

[11] Philip Axer, Maurice Sebastian, and Rolf Ernst. Probabilistic response time bound for can messages with arbitrary deadlines. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1114–1117. EDA Consortium, 2012.

[12] Philip Axer, Sophie Quinton, Moritz Neukirchner, Rolf Ernst, Björn Döbel, and Hermann Härtig. Response-time analysis of parallel fork-join workloads with real-time constraints. In *ECRTS*, 2013.

[13] Philip Axer, Daniel Thiele, and Rolf Ernst. Formal timing analysis of automatic repeat request for switched real-time networks. In *Proc. of SIES'14*, Pisa, Italy, June 2014.

[14] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. *Multiprocessor Scheduling for Real-Time Systems.* Springer International Publishing, 2015. doi: 10.1007/978-3-319-08696-5.

[15] Luca Benini and Giovanni De Micheli. Powering networks on chips: Energy-efficient and reliable interconnect design for SoCs. In *ISSS*, 2001. doi: 10.1145/500001.500009.

[16] Davide Bertozzi, Luca Benini, and Giovanni De Micheli. Error control schemes for on-chip communication links: the energy-reliability tradeoff. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(6), 2005.

[17] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[18] Nathan Binkert, Bradford Beckmann, Gabriel Black, et al. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[19] P. Bogdan, T. Dumitraş, and R. Marculescu. Stochastic communication: A new paradigm for fault-tolerant networks-on-chip. *VLSI design*, 2007, 2007.

[20] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10–16, Nov 2005. ISSN 0272-1732. doi: 10.1109/MM.2005.110.

[21] BroadR-Reach®:. BroadR-Reach® physical layer transceiver specification for automotive applications, v3.0. Broadcom Corporation, 2014.

[22] Alan Burns and Robert Davis. Mixed criticality systems - a review, 10th ed. *Department of Computer Science, University of York, Tech. Rep*, pages 1–72, January 2018.

[23] Alan Burns, James Harbin, and Leandro Soares Indrusiak. A wormhole noc protocol for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 184–195. IEEE, 2014.

[24] G. Buttazzo. *Hard Real-Time Computing Systems*. Springer US, New York, NY, USA, 2011. ISBN 978-1-4614-0676-1. doi: 10.1007/978-1-4614-0676-1.

[25] S. Chakraborty, S. Kunzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *2003 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 190–195, 2003. doi: 10.1109/DATE.2003.1253607.

[26] Cheng-Shang Chang. *Performance Guarantees in Communication Networks*. Springer-Verlag London, 2000. doi: 10.1007/978-1-4471-0459-9.

[27] Yung-Yuan Chen, Chung-Hsien Hsu, and Kuen-Long Leu. SoC-level risk assessment using FMEA approach in system design with SystemC. In *Industrial Embedded Systems, 2009. SIES '09. IEEE International Symposium on*, pages 82 –89, july 2009. doi: 10.1109/SIES.2009.5196199.

[28] C. Constantinescu. Trends and challenges in VLSI circuit reliability. *Micro, IEEE*, 23(4):14–19, July 2003. ISSN 0272-1732. doi: 10.1109/MM.2003.1225959.

[29] Rene L Cruz. A calculus for network delay, part i: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, 1991.

[30] Rene L Cruz. A calculus for network delay, part ii: Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, 1991.

[31] J. Diemer, D. Thiele, and R. Ernst. Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching. In *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pages 1–10, June 2012.

[32] Jonas Diemer and Rolf Ernst. Back Suction: Service Guarantees for Latency-Sensitive On-Chip Networks. In *The 4th ACM/IEEE International Symposium on Networks-on-Chip*, 2010.

[33] Jonas Diemer, Jonas Rox, Mircea Negrean, Steffen Stein, and Rolf Ernst. Real-Time Communication Analysis for Networks with Two-Stage Arbitration. In *EMSOFT'11*, October 2011.

[34] Jonas Diemer, Philip Axer, and Rolf Ernst. Compositional performance analysis in python with PyCPA. *Proc. of WATERS*, 2012.

[35] DO-254:. DO-254: Design assurance guidance for airborne electronic hardware. RTCA Incorporated, 2000.

[36] Björn Döbel and Hermann Härtig. Who watches the watchmen? protecting operating system reliability mechanisms. In *HotDep*, 2012.

[37] Björn Döbel, Hermann Härtig, and Michael Engel. Operating system support for redundant multithreading. In *Proceedings of the Tenth ACM International Conference on Embedded Software*, EMSOFT '12, pages 83–92, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1425-1. doi: 10.1145/2380356.2380375. URL `http://doi.acm.org/10.1145/2380356.2380375`.

[38] Björn Döbel, Hermann Härtig, and Michael Engel. Operating system support for redundant multithreading. In *Proc. of EMSOFT'12*, 2012.

[39] Tudor Dumitraş, Sam Kerner, and Radu Mărculescu. Towards on-chip fault-tolerant communication. In *Proceedings of the 2003 Asia*

and South Pacific Design Automation Conference, ASP-DAC '03, pages 225–232, New York, NY, USA, 2003. ACM. ISBN 0-7803-7660-9. doi: 10.1145/1119772.1119817.

[40] Guy Durrieu, Madeleine Faugere, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and Wolfgang Puffitsch. Predictable flight management system implementation on a multicore processor. In *Proc. of ERTS'14*, 2014.

[41] Mojtaba Ebrahimi, Adrian Evans, Mehdi B Tahoori, Enrico Costenaro, Dan Alexandrescu, Vikas Chandra, and Razi Seyyedi. Comprehensive analysis of sequential and combinational soft errors in an embedded processor. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1586–1599, 2015.

[42] L. Ecco and R. Ernst. Tackling the bus turnaround overhead in real-time sdram controllers. *IEEE Transactions on Computers*, 66 (11):1961–1974, Nov 2017. ISSN 0018-9340. doi: 10.1109/TC.2017.2714672.

[43] A. Ejlali, B.M. Al-Hashimi, P. Rosinger, S.G. Miremadi, and L. Benini. Performability/energy tradeoff in error-control schemes for on-chip networks. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 18(1):1 –14, jan. 2010. ISSN 1063-8210. doi: 10.1109/TVLSI.2008.2000994.

[44] EMC2 Project Consortium. D8.17 – safety aspects of hybrid avionic integrated architecture. Website: https://www.artemis-emc2.eu/, 2016.

[45] Michael Engel and Björn Döbel. The reliable computing base-a paradigm for software-based reliability. In *GI-Jahrestagung*, pages 480–493, 2012.

[46] R. Ernst and M. Di Natale. Mixed criticality systems x2014;a history of misconceptions? *IEEE Design Test*, 33(5):65–74, Oct 2016. ISSN 2168-2356. doi: 10.1109/MDAT.2016.2594790.

[47] Xiaocong Fan. *Real-Time Embedded Systems*. Newnes, Oxford, 2015. ISBN 978-0-12-801507-0. doi: https://doi.org/10.1016/B978-0-12-801507-0.09986-7. URL `https://www.sciencedirect.com/science/article/pii/B9780128015070099867`.

[48] Dror G. Feitelson and Larry Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16(4):306 – 318, 1992.

[49] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Minxuan Zhang, and Zuocheng Xing. Addressing transient and permanent faults in NoC with efficient fault-tolerant deflection router. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(6), 2013.

[50] David Fick, Andrew DeOrio, Jin Hu, Valeria Bertacco, David Blaauw, and Dennis Sylvester. Vicis: A reliable network for unreliable silicon. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 812–817, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-497-3. doi: 10.1145/1629911.1630119.

[51] Leandro S. Freitas, Eberle A. Rambo, and Luiz C. V. dos Santos. On-the-fly verification of memory consistency with concurrent relaxed scoreboards. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 631–636, San Jose, CA, USA, 2013. EDA Consortium. ISBN 978-1-4503-2153-2. URL `http://dl.acm.org/citation.cfm?id=2485288.2485444`.

[52] Rémi Gaillard. *Single Event Effects: Mechanisms and Classification*, pages 27–54. Springer US, Boston, MA, 2011. ISBN 978-1-4419-6993-4. doi: 10.1007/978-1-4419-6993-4_2. URL `https://doi.org/10.1007/978-1-4419-6993-4_2`.

[53] Joël Goossens and Vandy Berten. Gang ftp scheduling of periodic and parallel rigid real-time tasks. *arXiv preprint arXiv:1006.2617*, 2010.

[54] Kees Goossens, Arnaldo Azevedo, Karthik Chandrasekar, Manil Dev Gomony, Sven Goossens, Martijn Koedam, Yonghui Li, Davit Mirzoyan, Anca Molnos, Ashkan Beyranvand Nejad, Andrew Nelson, and Shubhendu Sinha. Virtual Execution Platforms for Mixed-time-criticality Systems: The CompSOC Architecture and Design Flow. *SIGBED Rev.*, 10(3):23–34, October 2013. ISSN 1551-3688. doi: 10.1145/2544350.2544353. URL `http://doi.acm.org/10.1145/2544350.2544353`.

[55] Jim Gray and Andreas Reuter. *Transaction processing: concepts and techniques*. Elsevier, 1992.

[56] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *WWC-4. 2001*, Dec 2001.

[57] Zain A. H. Hammadeh, Sophie Quinton, and Rolf Ernst. Extending typical worst-case analysis using response-time dependencies to bound deadline misses. In *Proc. of EMSOFT'14*, October 2014.

[58] Lance Hammond, Vicky Wong, Mike Chen, Brian D. Carlstrom, John D. Davis, Ben Hertzberg, Manohar K. Prabhu, Honggo Wijaya, Christos Kozyrakis, and Kunle Olukotun. Transactional memory coherence and consistency. *SIGARCH Comput. Archit. News*, 32(2): 102–, March 2004. ISSN 0163-5964. doi: 10.1145/1028176.1006711. URL `http://doi.acm.org/10.1145/1028176.1006711`.

[59] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, and Hiroaki Takada. Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis. *Journal of Information Processing*, 17:242–254, 2009. doi: 10.2197/ipsjjip.17.242.

[60] T. Harris, A. Cristal, O. S. Unsal, E. Ayguade, F. Gagliardi, B. Smith, and M. Valero. Transactional memory: An overview. *IEEE Micro*, 27(3):8–29, May 2007. ISSN 0272-1732. doi: 10.1109/MM.2007.63.

[61] Tino Heijmen. *Soft Errors from Space to Ground: Historical Overview, Empirical Evidence, and Future Trends*, pages 1–25. Springer US, 2011. ISBN 978-1-4419-6993-4.

[62] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System Level Performance Analysis–the SymTA/S Approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2), 2005.

[63] J. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. Härtig, L. Hedrich, A. Herkersdorf, R. Kapitza, D. Lohmann, P. Marwedel, M. Platzner, W. Rosenstiel, U. Schlichtmann, O. Spinczyk, M. Tahoori, J. Teich, N. When, and H. J. Wunderlich. Design and architectures for dependable embedded systems. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011 Proceedings of the 9th International Conference on*, pages 69–78, Oct 2011.

[64] Andreas Herkersdorf et al. Resilience Articulation Point (RAP): Cross-layer dependability modeling for nanometer system-on-chip resilience. *Microelectronics Reliability*, 54(6–7):1066–1074, 2014.

[65] Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. *SIGARCH Comput. Archit. News*, 21(2):289–300, May 1993. ISSN 0163-5964. doi: 10.1145/173682.165164. URL `http://doi.acm.org/10.1145/173682.165164`.

[66] M. Hoffmann, F. Lukas, C. Dietrich, and D. Lohmann. dosek: the design and implementation of a dependability-oriented static embedded kernel. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 259–270, April 2015. doi: 10.1109/RTAS.2015.7108449.

[67] Arnljot Høyland and Marvin Rausand. *System reliability theory: models and statistical methods*, volume 420. John Wiley & Sons, 2009.

[68] IEC 60812:. IEC 60812: Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA). International Electrotechnical Commission, 2006.

[69] IEC 61508:. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, ed.2.0. International Electrotechnical Commission, 2010.

[70] Leandro Soares Indrusiak, James Harbin, and Alan Burns. Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip. In *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, pages 47–56. IEEE, 2015.

[71] Leandro Soares Indrusiak, Alan Burns, and Borislav Nikolic. Buffer-aware bounds to multi-point progressive blocking in priority-preemptive NoCs, 2018. URL `http://eprints.whiterose.ac.uk/124747/`.

[72] ISO 26262:. ISO 26262: Road vehicles – functional safety. International Standards Organization, 2018.

[73] Yuming Jiang and Yong Liu. *Stochastic network calculus*, volume 1. Springer, 2008.

[74] T. Kadeed, E. A. Rambo, and R. Ernst. Power and area evaluation of a fault-tolerant network-on-chip. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pages 190–195, Sept 2017. doi: 10.1109/SOCC.2017.8226034.

[75] Hubert Kaeslin. *Digital integrated circuit design: from VLSI architectures to CMOS fabrication*. Cambridge University Press, 2008.

[76] Robert Kaiser and Stephan Wagner. Evolution of the PikeOS microkernel. In *First International Workshop on Microkernels for Embedded Systems*, 2007.

[77] Kalray. Mppa many-core family. `http://www.kalrayinc.com/kalray/products/#processors`, 2017. [Online; accessed 11-August-2017].

[78] Shinpei Kato and Yutaka Ishikawa. Gang EDF scheduling of parallel task systems. In *Proc. of RTSS'09*, 2009.

[79] Abbas Eslami Kiasari, Axel Jantsch, and Zhonghai Lu. Mathematical formalisms for performance evaluation of Networks-on-Chip. *ACM Computing Surveys*, 45(3), 2013.

[80] Cédric Killian, Camel Tanougast, Fabrice Monteiro, and Abbas Dandache. Smart reliable Network-on-Chip. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 22, 2014.

[81] Jongman Kim, Dongkook Park, C. Nicopoulos, N. Vijaykrishnan, and C.R. Das. Design and analysis of an NoC architecture from performance, reliability and energy perspective. In *ANCS*, 2005.

[82] A. Kohler, G. Schley, and M. Radetzki. Fault tolerant network on chip switching with graceful performance degradation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(6), 2010. ISSN 0278-0070. doi: 10.1109/TCAD.2010.2048399.

[83] Philip Koopman and Tridib Chakravarty. Cyclic redundancy code (CRC) polynomial selection for embedded networks. In *Proc. of DSN'04*, 2004.

[84] Adam Kostrzewa, Selma Saidi, Leonardo Ecco, and Rolf Ernst. Dynamic admission control for real-time networks-on-chips. In *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, pages 719–724. IEEE, 2016.

[85] H. T. Kung and John T. Robinson. On optimistic methods for concurrency control. *ACM Trans. Database Syst.*, 6(2):213–226, June 1981. ISSN 0362-5915. doi: 10.1145/319566.319567. URL `http://doi.acm.org/10.1145/319566.319567`.

[86] Dmitrii Kuvaiskii, Rasha Faqeh, Pramod Bhatotia, Pascal Felber, and Christof Fetzer. HAFT: hardware-assisted fault tolerance. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 25. ACM, 2016.

[87] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer Science & Business Media, 2001.

[88] S. Lee, I. Kim, S. Ha, C. s. Yu, J. Noh, S. Pae, and J. Park. Radiation-induced soft error rate analyses for 14 nm finfet sram devices. In *2015 IEEE International Reliability Physics Symposium*, pages 4B.1.1–4B.1.4, April 2015. doi: 10.1109/IRPS.2015.7112728.

[89] JP Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th Real-Time Systems Symposium*, 1990.

[90] Alberto Leon-Garcia and Indra Widjaja. *Communication Networks: Fundamental Concepts and Key Architectures, 2nd Edition.* McGraw-Hill, 2003.

[91] J.L. Leray. Effects of atmospheric neutrons on devices, at sea level and in avionics embedded systems. *Microelectronics Reliability*, 47 (9-11), 2007. 18th European Symposium on Reliability of Electron Devices, Failure Physics and Analysis.

[92] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[93] Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000. ISBN 0130996513.

[94] D. Manfield and P. Tran-Gia. Analysis of a finite storage system with batch input arising out of message packetization. *IEEE Transactions on Communications*, 30(3):456–463, Mar 1982. ISSN 0090-6778. doi: 10.1109/TCOM.1982.1095495.

[95] R. Marculescu, U.Y. Ogras, L.S. Peh, N.E. Jerger, and Y. Hoskote. Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(1):3, 2009.

[96] R. Mariani, G. Boschi, and F. Colucci. Using an innovative SoC-level FMEA methodology to design in compliance with IEC61508. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1 –6, april 2007. doi: 10.1109/DATE.2007.364641.

[97] P. Marwedel. *Embedded System Design*. Springer International Publishing, Cham, Switzerland, 2018. ISBN 978-3-319-56045-8. doi: 10.1007/978-3-319-56045-8.

[98] K. Matheus and T. Königseder. *Automotive Ethernet*. Cambridge University Press, New York, NY, USA, 2nd edition, 2017. ISBN 9781316872871.

[99] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking (TON)*, 7(2):188–201, 1999.

[100] Mellanox. Multicore processors overview. `http://www.mellanox.com/page/multi_core_overview?mtag=multi_core_overview`, 2017. [Online; accessed 11-August-2017].

[101] *Questa SIM User's Manual*. Mentor Graphics Corporation, Wilsonville, OR, 2016.

[102] J. Eliot B. Moss and Antony L. Hosking. Nested transactional memory: Model and architecture sketches. *Science of Computer Programming*, 63(2):186 – 201, 2006. ISSN 0167-6423. doi: https://doi.org/10.1016/j.scico.2006.05.010. URL `http://www.sciencedirect.com/science/article/pii/S0167642306001183`. Special issue on synchronization and concurrency in object-oriented languages.

[103] Boris Motruk, Jonas Diemer, Rainer Buchty, Rolf Ernst, and Mladen Berekovic. IDAMC: A Many-Core Platform with Run-Time Monitoring for Mixed-Criticality. In *Proc. of HASE'12*, 2012. doi: 10.1109/HASE.2012.19.

[104] S. Murali, T. Theocharides, N. Vijaykrishnan, M.J. Irwin, L. Benini, and G. De Micheli. Analysis of error recovery schemes for networks on chips. *Design Test of Computers, IEEE*, 22(5):434 – 442, sept.-oct. 2005. ISSN 0740-7475. doi: 10.1109/MDT.2005.104.

[105] Srinivasan Murali, David Atienza, Luca Benini, and Giovanni De Micheli. A method for routing packets across multiple paths in NoCs with in-order delivery and fault-tolerance gaurantees. *VLSi DeSign*, 2007.

[106] Takuya Nakaike, Rei Odaira, Matthew Gaudet, Maged M. Michael, and Hisanobu Tomari. Quantitative comparison of hardware transactional memory for blue gene/q, zenterprise ec12, intel core, and power8. *SIGARCH Comput. Archit. News*, 43(3):144–157, June 2015. ISSN 0163-5964. doi: 10.1145/2872887.2750403. URL `http://doi.acm.org/10.1145/2872887.2750403`.

[107] Netspeed. Netspeed Gemini SoC Interconnect. `http://netspeedsystems.com/products/gemini/`, 2018. [Online; accessed 04-May-2018].

[108] Netspeed. Netspeed Orion SoC Interconnect. `http://netspeedsystems.com/products/orion/`, 2018. [Online; accessed 04-May-2018].

[109] M. Neukirchner. *Establishing Sufficient Temporal Independence Efficiently: A Monitoring approach.* PhD thesis, TU Braunschweig, 2014.

[110] NXP MPC577xK Ultra-Reliable MCU Family. [online]. Available: http://www.nxp.com/assets/documents/data/en/fact-sheets/MPC577xKFS.pdf, 2017.

[111] Fabian Oboril and Mehdi B. Tahoori. Exploiting instruction set encoding for aging-aware microprocessor design. *ACM Trans. Des. Autom. Electron. Syst.*, 21(1):5:1–5:26, December 2015.

[112] M. Ottavi, S. Pontarelli, D. Gizopoulos, C. Bolchini, M. K. Michael, L. Anghel, M. Tahoori, A. Paschalis, P. Reviriego, O. Bringmann, V. Izosimov, H. Manhaeve, C. Strydis, and S. Hamdioui. Dependable multicore architectures at nanoscale: The view from Europe. *IEEE Design Test*, 32(2):17–28, April 2015. ISSN 2168-2356. doi: 10.1109/MDAT.2014.2359572.

[113] John K Ousterhout. Scheduling techniques for concurrent systems. In *ICDCS*, volume 82, pages 22–30, 1982.

[114] Zaher Owda, Moisés Urbina, Roman Obermaisser, and Mohammed Abuteir. Hierarchical transactional memory protocol for distributed mixed-criticality embedded systems. In *Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), 2016 IEEE 14th Intl C*, pages 334–343. IEEE, 2016.

[115] J. C. Palencia and M. Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proc. of RTSS'98*, 1998.

[116] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C.R. Das. Exploring fault-tolerant Network-on-Chip architectures. In *Proc. of DSN'06*, 2006.

[117] Rodolfo Pellizzoni, Emiliano Betti, Stanley Bak, Gang Yao, John Criswell, Marco Caccamo, and Russell Kegley. A predictable execution model for COTS-based embedded systems. In *Proc. of RTAS'11*, 2011.

[118] M. Pirretti, G.M. Link, R.R. Brooks, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin. Fault tolerant algorithms for Network-on-Chip interconnect. In *VLSI, 2004. Proceedings. IEEE Computer society Annual Symposium on*, pages 46 – 51, feb. 2004. doi: 10.1109/ISVLSI.2004.1339507.

[119] Yue Qian, Zhonghai Lu, and Wenhua Dou. Analysis of worst-case delay bounds for best-effort communication in wormhole networks on chip. In *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pages 44–53, May 2009. doi: 10.1109/NOCS.2009.5071444.

[120] Yue Qian, Zhonghai Lu, and Wenhua Dou. Analysis of worst-case delay bounds for on-chip packet-switching networks. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29 (5):802–815, May 2010. ISSN 0278-0070. doi: 10.1109/TCAD.2010.2043572.

[121] Martin Radetzki, Chaochao Feng, Xueqian Zhao, and Axel Jantsch. Methods for fault tolerance in Networks-on-Chip. *ACM Comput. Surv*, 46(1):8:1–8:38, 2013.

[122] E. A. Rambo, O. P. Henschel, and L. C. V. dos Santos. Automatic generation of memory consistency tests for chip multiprocessing. In *2011 18th IEEE International Conference on Electronics, Circuits, and Systems*, pages 542–545, Dec 2011. doi: 10.1109/ICECS.2011.6122332.

[123] E. A. Rambo, O. P. Henschel, and L. C. V. dos Santos. On esl verification of memory consistency for system-on-chip multiprocessing. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 9–14, March 2012. doi: 10.1109/DATE.2012.6176424.

[124] E. A. Rambo, L. Ahrendts, and J. Diemer. FMEA of the IDAMC NoC. Technical report, Institute of Computer and Network Engineering – TU Braunschweig, 2013.

[125] Eberle A. Rambo and Rolf Ernst. Worst-case communication time analysis of networks-on-chip with shared virtual channels. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015.

[126] Eberle A Rambo and Rolf Ernst. Providing flexible and reliable on-chip network communication with real-time constraints. In *1st International Workshop on Resiliency in Embedded Electronic Systems (REES)*, 2015.

[127] Eberle A. Rambo and Rolf Ernst. Replica-Aware Co-Scheduling for Mixed-Criticality. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, volume 76, pages 20:1–20:20, 2017. ISBN 978-3-95977-037-8. doi: 10.4230/LIPIcs.ECRTS.2017.20.

[128] Eberle A Rambo, Alexander Tschiene, Jonas Diemer, Leonie Ahrendts, and Rolf Ernst. Failure analysis of a Network-on-Chip

for real-time mixed-critical systems. In *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014.

[129] Eberle A Rambo, Alexander Tschiene, Jonas Diemer, Leonie Ahrendts, and Rolf Ernst. FMEA-Based Analysis of a Network-on-Chip for Mixed-Critical Systems. In *2014 Eighth IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*, pages 33–40, Sept 2014. doi: 10.1109/NOCS.2014.7008759.

[130] Eberle A. Rambo, Selma Saidi, and Rolf Ernst. Providing formal latency guarantees for ARQ-based protocols in Networks-on-Chip. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016.

[131] Eberle A Rambo, Christoph Seitz, Selma Saidi, and Rolf Ernst. Bridging the gap between resilient networks-on-chip and real-time systems. *IEEE Transactions on Emerging Topics in Computing*, 2017. URL `http://doi.org/10.1109/TETC.2017.2736783`.

[132] Eberle A. Rambo, Christoph Seitz, Selma Saidi, and Rolf Ernst. Designing networks-on-chip for high assurance real-time systems. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 185–194, Jan 2017. doi: 10.1109/PRDC.2017.32.

[133] Eberle A Rambo, Yunsheng Shang, and Rolf Ernst. Providing integrity in real-time networks-on-chip. *Submitted to IEEE Transactions on Very Large Scale Integration Systems*, 2018.

[134] Maurizio Rebaudengo, Matteo Sonza Reorda, and Massimo Violante. *Software-Level Soft-Error Mitigation Techniques*, pages 253–285. Springer US, Boston, MA, 2011. ISBN 978-1-4419-6993-4. doi: 10.1007/978-1-4419-6993-4_9. URL `https://doi.org/10.1007/978-1-4419-6993-4_9`.

[135] K. Richter. *Compositional Scheduling Analysis Using Standard Event Models*. PhD thesis, TU Braunschweig, 2005.

[136] Thiago Santini, Christoph Borchert, Christian Dietrich, Horst Schirmeier, Martin Hoffmann, Olaf Spinczyk, Daniel Lohmann,

Flávio Rech Wagner, and Paolo Rech. *Effectiveness of Software-Based Hardening for Radiation-Induced Soft Errors in Real-Time Operating Systems*, pages 3–15. Springer International Publishing, Cham, 2017. ISBN 978-3-319-54999-6. doi: 10.1007/978-3-319-54999-6_1. URL https://doi.org/10.1007/978-3-319-54999-6_1.

[137] Simon Schliecker, Jonas Rox, Rafik Henia, Razvan Racu, Arne Hamann, and Rolf Ernst. *Formal performance analysis for real-time heterogeneous embedded systems*, pages 57–92. CRC Press, 2009. ISBN 9781420067842.

[138] Andreas Schranzhofer, Jian-Jia Chen, and Lothar Thiele. Timing analysis for TDMA arbitration in resource sharing systems. In *Proc. of RTAS'10*, 2010.

[139] Simon Schuster, Peter Ulbrich, Isabella Stilkerich, Christian Dietrich, and Wolfgang Schröder-Preikschat. Demystifying soft-error mitigation by control-flow checking – a new perspective on its effectiveness. *ACM Trans. Embed. Comput. Syst.*, 16(5s):180:1–180:19, September 2017. ISSN 1539-9087. doi: 10.1145/3126503. URL http://doi.acm.org/10.1145/3126503.

[140] S. Shamshiri, A.-A. Ghofrani, and Kwang-Ting Cheng. End-to-end error correction and online diagnosis for on-chip networks. In *Test Conference (ITC), 2011 IEEE International*, pages 1–10, Sept 2011. doi: 10.1109/TEST.2011.6139156.

[141] Jian Shi, Shaoping Wang, and Kang Wang. Challenges and evaluation method in network performability analysis. In *Robotics, Automation and Mechatronics (RAM), 2011 IEEE Conference on*, pages 96 –101, sept. 2011. doi: 10.1109/RAMECH.2011.6070463.

[142] Zheng Shi and Alan Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '08, pages 161–170, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3098-7.

[143] Sonics. Sonics Trusted NoC Products. https://sonicsinc.com/on-chip-networks/, 2018. [Online; accessed 04-May-2018].

[144] Tobias Stumpf. How to protect the protector. In *Proceedings of the th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'), Student Forum. IEEE Computer Society Press*, June 2015.

[145] *Design Compiler User Guide*. Synopsys Inc., Mountain View, CA, 2017.

[146] *PrimeTime User Guide*. Synopsys Inc., Mountain View, CA, 2017.

[147] A.S. Tanenbaum and D. Wetherall. *Computer Networks*. Pearson Prentice Hall, 2011. ISBN 9780132126953.

[148] Daniel Thiele, Jonas Diemer, Philip Axer, Rolf Ernst, and Jan Seyler. Improved formal worst-case timing analysis of weighted round robin scheduling for ethernet. In *In Proc. of CODES+ISSS*, Montreal, Canada, sep 2013.

[149] K.W. Tindell, A. Burns, and A.J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2), 1994. ISSN 0922-6443.

[150] Sebastian Tobuschat, Philip Axer, Rolf Ernst, and Jonas Diemer. IDAMC: A NoC for mixed criticality systems. In *Proc. of RTCSA'13*, 2013.

[151] Wen-Chung Tsai, Deng-Yuan Zheng, Sao-Jie Chen, and Yu-Hen Hu. A fault-tolerant noc scheme using bidirectional channel. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 918–923, 2011.

[152] Erik B Van Der Tol and Egbert G Jaspers. Mapping of MPEG-4 decoding on a flexible architecture platform. In *Electronic Imaging 2002*. International Society for Optics and Photonics, 2001.

[153] Andras Varga and Rudolf Hornig. An overview of the OMNeT++ simulation environment. In *1st SIMUTools 2008*, 2008.

[154] András Varga et al. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, volume 9, page 185, 2001.

[155] P. Vijaya Laxmi and U.C. Gupta. Analysis of finite-buffer multi-server queues with group arrivals: Gix/m/c/n. *Queueing Systems*, 36(1):125–140, Nov 2000. ISSN 1572-9443. doi: 10.1023/A:1019179119172. URL `https://doi.org/10.1023/A:1019179119172`.

[156] *Xilinx Power Tools Tutorial*. Xilinx Inc., San Jose, CA, 2013.

[157] *UG901: Vivado Design Suite User Guide: Synthesis*. Xilinx Inc., San Jose, CA, 2017.

[158] Q. Xiong, F. Wu, Z. Lu, and C. Xie. Extending real-time analysis for wormhole NoCs. *IEEE Transactions on Computers*, 66(9):1532–1546, Sept 2017. ISSN 0018-9340. doi: 10.1109/TC.2017.2686391.

[159] Gulay Yalcin, Osman Unsal, and Adrian Cristal. Faultm: error detection and recovery using hardware transactional memory. In *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 220–225. IEEE, 2013.

[160] Qiaoyan Yu, Bo Zhang, Yan Li, and P. Ampadu. Error control integration scheme for reliable NoC. In *ISCAS*, 2010.